



**Liguria
Digitale**

ITS
Accademia Digitale Liguria

ITS - ICT ACCADEMIA DIGITALE LIGURIA

*Corso tecnico superiore per i metodi e le tecnologie per la gestione di sistemi
software per l'industria 4.0*

SIMPLY CODE

LE SEMPLIFICAZIONI DEL SOFTWARE

A cura di:
Simone Davì

Codice Corso:
RLOF20ITS.3.2

Tutor Aziendale:
Matteo Scaldaferrri

INDICE

INTRODUZIONE.....	3
CAPITOLO 1 - LO STUDENTE.....	4
1.1 Presentazione Personale	
1.2 Obiettivo dell'Attività di Tirocinio	
CAPITOLO 2 - L'AZIENDA.....	5
2.1 Presentazione Azienda	
2.2 Mission e Vision Aziendale	
2.3 Organigramma e Dimensione	
CAPITOLO 3 - IL LAVORO ASSEGNATO.....	7
3.1 Presentazione delle Funzionalità	
3.2 Input e Analisi del Progetto	
CAPITOLO 4 - GLI STRUMENTI UTILIZZATI.....	10
4.1 Introduzione degli strumenti	
4.2 La vita del software all'interno dell'azienda	
4.3 Gli strumenti Utili	
4.4 Pattern e Codice	
4.5 La grafica e le sue regole	
CAPITOLO 5 - LO SVOLGIMENTO DELL'ATTIVITA'	24
5.1 Studio ed Apprendimento di nuovi Strumenti	
5.2 Modello Realizzativo	
5.3 Principali Risultati Raggiunti e Output	
CAPITOLO 6 - CONSIDERAZIONI FINALI.....	27
6.1 Crescita Personale e Consolidamento Tecnico	

INTRODUZIONE

In questo documento andremo ad affrontare la mia esperienza a Liguria Digitale nel periodo di Stage tra Luglio 2022 e Novembre 2022.

Gli argomenti trattati saranno legati allo sviluppo software con uno sguardo particolare per il **tema della “semplificazione”** in ambito di lavoro, interazione con il team, scrittura del software e altro.

Divisione del codice, Design Pattern, Norme di accessibilità informatica sono solo alcuni dei momenti in cui troviamo una *“ricerca di ottimizzazione”* nel mondo del software.

Spinto da un desiderio di **semplificazione** in collaborazione con un **ideale di aggiornamento**, ho posto il mio focus su qualcosa che può rendere l’ambiente lavorativo più sereno e la vita dei cittadini liguri più facile.

I miei compiti sono stati lo sviluppo di app per l’automatizzazione di processi aziendali comuni (A) e l’implementazione di funzioni per la modernizzazione della sanità digitale ligure (B).

- A. **EDAE Engine**, un’applicazione backend di generazione di JSON per la compilazione di schede per la sanità rivisitata per poter generare JSON attraverso l’utilizzo di servizi REST.
- B. **Il Teleconsulto Asincrono**, una web app che punta a semplificare il dialogo tra professionisti sanitari per venire incontro ai pazienti sia nelle tempistiche che nella scelta del trattamento.

Andremo poi ad analizzare il lavoro svolto, passando per la spiegazione degli strumenti e standard usati.

Concluderemo poi vedendo cosa questa esperienza abbia portato nel suo pacchetto di conoscenze ed esperienze e come mi è stata utile lavorativamente e umanamente.

Per necessità di privacy certe cose potrebbero essere state modificate o tralasciate nel rispetto del rapporto con l’azienda

CAPITOLO 1 - LO STUDENTE

1.1 Presentazione Personale

Simone Davì, nato il 20/01/2001 a Genova, mi sono diplomato in Elettronica ed Elettrotecnica articolazione **Automazione**, all'ITIS Italo Calvino Genova, nell'anno scolastico 2018/2019 con il voto di 89/100. Nel 19'/20' ho svolto un anno di Ingegneria Elettrica, lasciandola per spostare il mio focus sullo **sviluppo software**. Dall'anno 2020/2021 frequento l'**ITS-ICT Accademia Digitale Liguria** per approfondire, migliorare e consolidare le mie conoscenze nell'ambito informatico.

Ho cominciato a sviluppare la mia figura lavorativa dalla scuola, cominciando in ambiti di **domotica** e **elettronica**, spostandomi poi sull'**insegnamento** dell'informatica ai ragazzi, per poi concentrarmi sullo **sviluppo software**.

Ho partecipato a vari progetti partendo dal 2017 con il premio di miglior studente Ligure di domotica, vincendo uno stage nella sede di CAME s.p.a.. Ho cominciato ad insegnare robotica all'Istituto Comprensivo di Arenzano e successivamente anche informatica ed elettronica con Scuola di Robotica. Durante il corso l'ITS ho partecipato al progetto Glove, una collaborazione in ambito di realtà virtuale aptica con ETT s.p.a.

Ora sviluppo software principalmente web per Scuola di Robotica in vari progetti anche europei.



1.2 Obiettivo dell'Attività di Tirocinio

Ho scelto di svolgere il Tirocinio a Liguria Digitale poiché a seguito del primo periodo di Stage ho capito che l'ambiente era in sintonia con i miei bisogni e i miei interessi. Mi è stato richiesto di partecipare attivamente allo sviluppo di due applicazioni in ambito di sanità, per poter apprendere nuovi strumenti all'interno del mondo del software, come anche concetti di analisi e versatilità.

CAPITOLO 2 - L'AZIENDA - Liguria Digitale

2.1 Presentazione Azienda



Liguria Digitale nasce nel 2015 a seguito della trasformazione societaria da Datasiel S.p.A., azienda di consulenza informatica e software, a Liguria Digitale S.C.p.A.

Prende il nome corrente di Liguria Digitale S.p.A. solo a seguito della seconda trasformazione societaria nel gennaio 2017.

La Sede Legale si trova al Parco Scientifico e Tecnologico sulla collina degli **Erzelli** di Genova ed è il luogo dove si è svolto il periodo di Stage.

L'azienda si impegna a sviluppare la **strategia digitale della Regione Liguria** e degli enti soci per i cittadini, le imprese, i turisti e la pubblica amministrazione ligure, realizzando soluzioni informatiche di qualità e servizi di interesse generale.

Uno sguardo importante viene posto a temi come lo sviluppo di progetti **innovativi e tecnologicamente all'avanguardia**, per fornire a tutti servizi digitali e informazioni facilmente accessibili. **Data la ricerca costante di miglioramento e di sicurezza**, Liguria Digitale si impegna anche a garantire degli standard di continuità molto alti grazie anche all'utilizzo del data center al **WTC (World Trade Center)** e architetture cloud orientate ai servizi.



L'azienda rappresenta anche lo strumento operativo del SIIR, il **Sistema Informativo Regionale Integrato**, che consente di operare organicamente sul territorio, con qualità e riduzione dei costi, attraverso un uso condiviso delle tecnologie più avanzate, la razionalizzazione nell'acquisizione delle risorse e il contenimento della spesa pubblica.

2.2 Mission e Vision

"La Liguria è la regione di eccellenza dove è bello vivere, lavorare e trascorrere il proprio tempo libero."

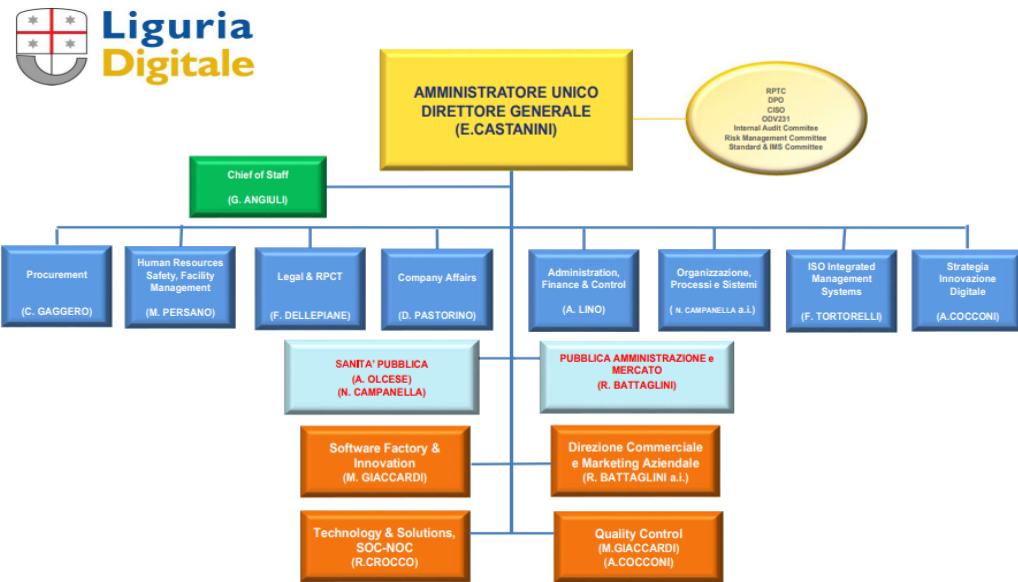
L'impegno di Liguria Digitale è quello di fornire un'infrastruttura digitale all'avanguardia diffusa e integrata su tutto il territorio ligure, in modo che il cittadino, il turista e l'impresa realizzino i propri obiettivi e possano usufruire di servizi accessibili e veloci.

Per rendere la nostra regione più coesa, più forte e più attrattiva, Liguria Digitale persegue i seguenti obiettivi:

- Gestire al completo l'infrastruttura digitale presente e futura per la pubblica amministrazione, al servizio di cittadini, turisti e imprese, in accordo con le direttive strategiche degli enti soci.
- Contribuire al miglioramento della qualità della vita, con servizi ad alto valore aggiunto.
- Favorire la standardizzazione e un uso condiviso delle tecnologie più avanzate, per un'evoluzione integrata e omogenea di un sistema informativo su tutto il territorio ligure.
- Favorire l'economia locale, il commercio e l'industria, incluse le attività portuali.

- Supportare la modernizzazione delle infrastrutture pubbliche e delle amministrazioni, avvicinandole ai cittadini, rendendole più semplici e accessibili, apprezzandone qualità e riduzione dei costi.
- Promuovere la Liguria nel mondo attraverso campagne di comunicazione e marketing digitale sviluppate e gestite per i nostri clienti e soci.

2.3 Organigramma e Dimensione



"Quest'azienda risulta tra le prime 100 della nostra regione in base al fatturato 2020: con il dato 2021, con un fatturato da 80 milioni di euro, in aumento del 15% rispetto all'anno precedente - aggiunge il Presidente della Regione Liguria, Giovanni Toti - Liguria Digitale sarebbe posizionata ancora meglio. Abbiamo consolidato le scelte strategiche, soprattutto quelle orientate alla gestione dell'emergenza sanitaria e della campagna vaccinale anti covid19 e quelle orientate al rafforzamento delle infrastrutture digitali e della cybersecurity".

liguriadigitale.it - "Approvato il bilancio 2021: fatturato in crescita del 15%"

La forza lavoro è rappresentata da 435 dipendenti interni, 495 in somministrazione, 170 clienti, 544 fornitori e 680 contratti/commesse per anno, che insieme a un fatturato di 80 milioni, porta l'azienda a riconfermare l'impegno su tutto il territorio ligure come Grande Impresa secondo gli standard europei.

CAPITOLO 3 - IL LAVORO ASSEGNATO

3.1 Presentazione delle Funzionalità

I principali compiti possono essere divisi in due:

- EDAE Engine, **L'automatizzazione di un processo** di compilazione di JSON precedentemente usato solo in ambito di sanità per la compilazione di referti medici specifici.
- Lo sviluppo dell'app di **Teleconsulto**. Il *teleconsulto sincrono* è la **condivisione in real time** di un consulto tra 2 (o più medici) su un paziente. Il *teleconsulto asincrono* non è **real time**: un medico (di solito il medico di medicina generale MMG) richiede il “consulto” (consulenza) ad un altro medico relativo ad un paziente. In un momento successivo, lo specialista invierà una risposta ed eventualmente potrà allegare della documentazione.

“EDAE”, ovvero **Elastic Data Access Engine**, è un’implementazione di backend proposta a seguito di uno studio di bisogni nell’ambito sanità, specialmente nel contesto della web app relativa alla gestione delle cure domiciliari, ramo della piattaforma software per la sanità, dove si ha la necessità che un assistente abilitato alla visita medica a domicilio abbia lo storico clinico delle visite del paziente assistito.

Obiettivi

- Gestire richieste eterogenee di «pagine standard» e «pagine custom» attraverso la stessa interfaccia REST
- Storicizzare e versionare i JSON (di template e data)
- Garantire maggiore resilience in caso di cambi improvvisi nel modello dati
- Garantire una maggiore disponibilità del dato anche in caso difficoltà a raggiungere il database o in caso di degrado delle performance di quest’ultimo

“Presentazione_Backend_EDAE” - Riccardo Riggi

La prima versione propone una risoluzione a un problema possibilmente comune con delle **specifiche però singole**, il che introduce la necessità di **“portabilità”** e il bisogno di standardizzare il modello di inserimento dati delocalizzandolo dalle viste di Database.

Il mio lavoro di automatizzazione è stato svolto al fine di creare un backend in **feature branch**, riutilizzabile ovunque sia necessaria una compilazione di JSON composti da template, con dati ricevuti **sia da Database che da servizi REST**, e output JSON fornito in servizio REST.

Nel caso del *Teleconsulto Asincrono* il mio ruolo è stato quello di implementare le funzioni necessarie per lo sviluppo della web app in **full stack**, ovvero sia a contatto con il frontend che con il backend.

Le implementazioni:

- Frontend
 - Pagine “**CRUD**” (Elementi di Inserimento, lettura , modifica, eliminazione)
 - Componenti Web
- Backend
 - Endpoint

3.2 Input e Analisi del Progetto

Per EDAE è corretto considerare come input non solo gli effettivi dati, e template ma anche la versione precedente del codice.

Riporto di seguito delle slide significative dell'analisi di progetto legata alla web app per le cure domiciliari:

- In origine la logica di BE presenta una divisione Standard / Custom la quale nonostante la parità di I / O utilizza due metodi diversi per la creazione dei JSON.
- Nella seconda e terza immagine vediamo in risalto il concetto di legame singolo con i database, dove non solo la divisione di dati, ma anche la gerarchia delle viste da ricercare è basata su una sequenza di tabelle e schede. Notiamo come la ripetizione dei nomi delle colonne non sia casuale, ma risulti standardizzata per una ricerca di sequenza in scala.
- Uno sguardo importante verrà posto nella sezione **4.4 Pattern e Codice** al concetto di **DAO**, Data Access Object, ovvero un pattern strutturale per la gestione della persistenza, e nel suo ruolo nella semplificazione del codice.
- Nell'analisi sono stati individuati i primi **miglioramenti e problematiche** nella necessità di mostrare i dati, con la criticità che la differenziazione di JSON Standard e Custom impedirebbe di nuovo di fare una classe comune per la riproduzione a video, costringendo degli algoritmi unici localizzati.

La logica di backend

1

Il motore standard accede al dato attraverso uno strato di DAO parametrizzabile per:

- Nome dell'oggetto da interrogare (vista in questo caso)
- Eventuali filtri da impostare per selezionare l'insieme richiesto
- Elenco dei campi necessari alla restituzione delle informazioni esaustive richieste

La logica di backend – Da pagina a tabella

2

Esiste una tavola di lookup che, dato il path di una pagina/scheda, restituisce il nome della tavola/vista dalla quale recuperare le informazioni

PATH	NOME_VISTA	GRADO_COMPLESSITA
1 pic	SCHEDA_PRESA_IN_CARICO	STANDARD
2 cirs	SCHEDA_CIRS	STANDARD
3 iadl	SCHEDA_IADL	STANDARD
4 agenda-accessi	DIARIO_ACCESSI	CUSTOM
5 attivita-riabilitativa-e-riattivanti	SCHEDA_ATIV_RIABIL	STANDARD

La logica di backend - Filtering

3

Esiste una tavola che, dato il nome della vista precedentemente cercata, restituisce l'elenco dei campi necessari da mettere nelle condizioni di WHERE per estrarre il dato corretto

NOME_VISTA	NOME_PARAMETRO	TIPO_PARAMETRO	ORDINE	NOME_CAMPO_JSON
1 SCHEDA_PRESA_IN_CARICO	IDPIC	TESTO	1	idPic
2 SCHEDA_CIRS	IDPIC	TESTO	1	idPic
3 SCHEDA_IADL	IDPIC	TESTO	1	idPic
4 SCHEDA_ATIV_RIABIL	IDPIC	TESTO	1	idPic

Riguardo al Teleconsulto Asincrono possiamo cominciare dall'**Analisi Funzionale**, la quale esamina necessità, obiettivi e risultati del progetto.

Partendo dallo scopo del progetto “**Tigullio come Luogo di Salute**”, ovvero **l'identificazione di un modello condiviso per l'erogazione delle cure domiciliari**, che sfrutti al meglio le possibilità offerte dalle nuove tecnologie (telemedicina, domotica, digitalizzazione), vediamo come il progresso spesso si avvicini a un concetto di “comunità” per la semplificazione dei processi.

Facendo un passo avanti alla vera e propria Web App possiamo cominciare ad analizzarne gli obiettivi.

2.1. Obiettivi

I Medici di Medicina Generale e i medici afferenti a centri di assistenza “spoke” hanno l'esigenza di chiedere consulti, spesso nella modalità asincrona che non richiede accordi per un appuntamento tra professionisti in un preciso orario; non possiedono uno strumento dedicato allo scopo per indirizzare quesiti a specialisti o gruppo di medici “hub” relativi una persona che hanno in assistenza.

Analisi_Funzionale_Teleconsulto_Async L.Tranzatto

L'analisi non si ferma solo all'astrazione di un problema e degli obiettivi, ma comincia anche a dare le **prime regolamentazioni** per gli sviluppatori, per permettere a un gruppo per quanto eterogeneo nelle mansioni e conoscenze, di potersi relazionare e interagire a pieno.

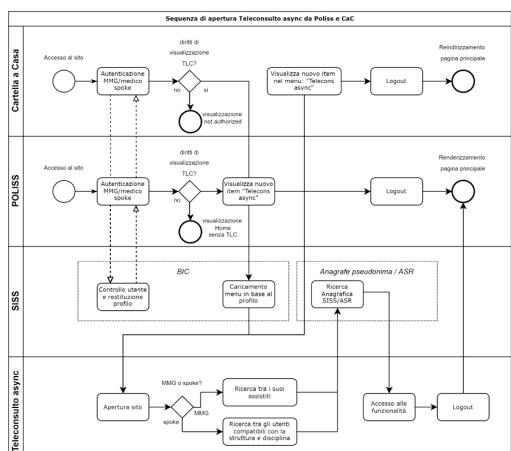
- Codice:
 - Dichiarazione delle caratteristiche degli utenti quali la divisione in REQ, cioè richiedenti di Teleconsulto, e RES, coloro abilitati a rispondere alle richieste.
 - Definizione dei campi. Non è necessaria solo l'identificazione delle figure di richiesta e risposta ma anche le loro particolarità quali per esempio la disciplina di specializzazione definita da campo COD_SPECIALITA, tabella T_SPECIALITA dello schema TELECONS
- Requisiti Funzionali:
 - Modalità di Autenticazione e accesso per operatore
 - Formazione Menù del sito
 - Gestione ricerche file / utente
 - Gestione Sequenza Accessi ¹
 - Gestione Upload File in teleconsulto (Gerarchia Path e Tipologia file)
 - Funzionalità legate alla versione in produzione
- Requisiti Visivi:
 - Bozza dashboard generale ²
 - Definizione sistema “semaforico” per l'individuazione degli stati del Teleconsulto

Aperto in attesa In elaborazione Concluso con risposta In Bozza

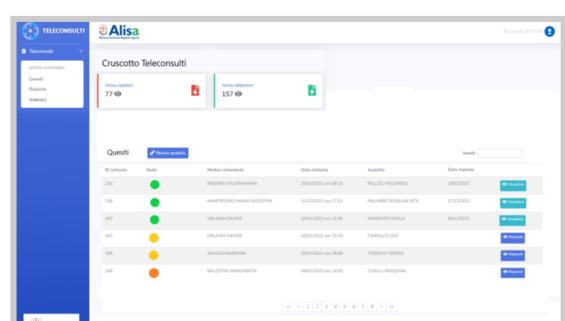
- Definizione Oggetti a servizio dell'utente:
 1. Pagine **CRUD Richiesta**
 2. Pagine **CRUD Risposta**

CRUD = Create, Read, Update, Delete

¹ Riguardo alla gestione sequenza accessi abbiamo inoltre la schematizzazione della sequenza di apertura. Notiamo l'utilizzo dei **diagrammi di flusso**, utili per una rappresentazione sequenziale temporale di come dovranno essere svolte le operazioni.



² La bozza della pagina del teleconsulto



CAPITOLO 4 - GLI STRUMENTI UTILIZZATI

4.1 Introduzione agli strumenti

EDAE	TELECONSULTO ASINCRONO
<ul style="list-style-type: none">• Red Hat Wildfly 14 (Server)• JDK 1.8 (Ambiente di Sviluppo)• Insomnia (Test servizi Rest)• Oracle SQLDeveloper (Database)	<ul style="list-style-type: none">• Apache Tomcat 9 (Server)• Open JDK 11 (Ambiente di Sviluppo)• Microsoft Visual Studio Code (IDE Sviluppo Frontend)• SpringBoot (Framework BE)• MyBatis (Framework Persistenza)• JavaScript - TypeScript (Linguaggio programmazione Frontend)• Meta React (Libreria FE)• WAVE (Strumento di valutazione accessibilità web)• Atlassian Jira (Product & Issue Tracker)• Jenkins (Supporto CI / CD)
COMUNI	
<ul style="list-style-type: none">• Eclipse (IDE Sviluppo BE)• Java (Linguaggio di programmazione BE)• Git & GitHub (Gestione Codice)	

WildFly e Apache Tomcat sono gli strumenti utilizzati per la fruizione delle piattaforme software di utilizzo dei programmi in fase di sviluppo. WildFly è un **application server** realizzato in Java che implementa a pieno le **specifiche JavaEE**, al contrario di Tomcat il quale utilizza solo parte delle specifiche, cioè **JavaServer Pages (JSP)** e **servlet**, e per questo viene considerato un **web server**.



JDK o Java Development Kit, è l'**ambiente di sviluppo** più utilizzato per la programmazione Java. Si tratta di un insieme di componenti indispensabili e utili per la programmazione. All'interno di alcuni ambienti di sviluppo, viene agganciato anche il **JRE**, **Java Runtime Environment**, ovvero l'**ambiente di esecuzione Java** per applicazioni più complesse, esempio quelle web.



Insomnia è uno strumento di **test e validazione di applicazioni RESTful** (vai a 4.4 Standard, Pattern e Codice) che ci permette di utilizzare i metodi **HTTP**

GET POST PUT PATCH DELETE OPTIONS HEAD

per gestire le richieste e le risposte da parte dei servizi sviluppati, controllare **Header**, **Query** e possibili variabili **JSON**.



Visual Studio Code e **Eclipse** sono gli IDE, **Ambienti di Sviluppo Integrato** in italiano, utilizzati per lo sviluppo del codice durante il project work. Sono entrambi **multi linguaggio** ma mentre Eclipse è anche multipiattaforma e open source, VSC è gestito da Microsoft per Windows, nonché specialmente usato, come in questo caso, per lo sviluppo frontend. Eclipse al contrario, data anche la sua nascita come only-Java-IDE poi ampliata anche grazie ai plugin scaricabili dall’“Eclipse Marketplace”, si utilizza maggiormente per il backend.



Java e **JavaScript** sono i **linguaggi di programmazione** utilizzati. **Java** è stato usato per tutta la parte relativa a EDAE e insieme a Spring Boot per la parte **backend** del Teleconsulto Asincrono. **JavaScript** è stato utilizzato invece per la parte di **frontend** legato alla libreria di React, viene utilizzata per la creazione di **User Interfaces**. **TypeScript** invece rappresenta un’**estensione di JS** basata sullo standard ECMAScript 6 o ES6, questo linguaggio rispetto all’originale presenta una tipizzazione maggiore consentendo di descrivere la forma di un oggetto, documentandolo meglio e consentendo di verificare che il codice funzioni correttamente.



React (Libreria) e **Spring Boot** sono **framework**, cioè **un’architettura logica di supporto** dove si implementano nel programma una serie di classi astratte e librerie per implementare o modificare delle funzioni. Tra di loro sono molto differenti anche solo nello scopo, mentre **React** è una libreria utilizzata in collaborazione con JavaScript per uno **sviluppo frontend** delle UI, **Spring Boot**, essendo un **framework backend**, si interfaccia con i servizi implementati con Java per lo sviluppo del Teleconsulto.

MyBatis è stato l’ultimo framework utilizzato in **Java** e differisce dai precedenti per utilizzo dato il suo ruolo attivo nella **persistenza**. È stato usato nell’applicazione di gestione delle visite a domicilio per il collegamento con i database, portando a una semplificazione del codice data la semplicità nell’utilizzo dei mapper per le chiamate alle tabelle.



WAVE Evaluation Tool è uno strumento di **valutazione grafica** di pagine web, può essere aggiunto a Chrome come estensione e serve a individuare i **possibili errori grafici** commessi, i quali potrebbero portare utenti con disabilità visive o daltonismo a non poter usufruire del sito.

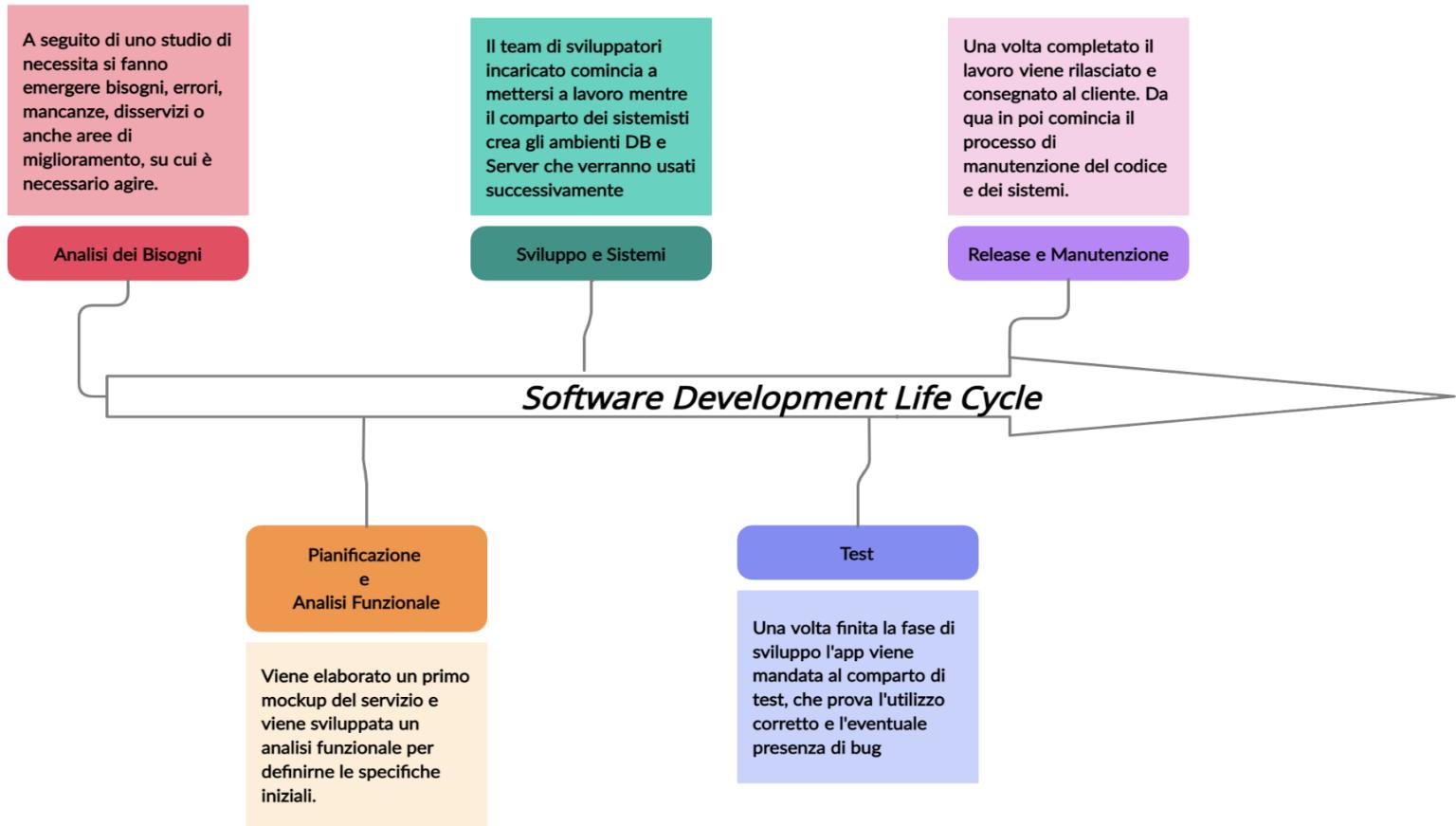
Jira è il software di **gestione delle segnalazioni e della divisione del lavoro**, mentre **Jenkins** è uno strumento che fornisce servizi di **integrazione continua**. Infine **GitTea** e **GitHub Desktop** sono stati gli strumenti di **Hosting e gestione del codice**. Data la loro importanza nell’argomento Semplificazione, approfondiremo i loro ruoli al punto **4.3 Gli strumenti Utili**.



Jira Software



4.2 La vita del software all'interno dell'azienda



4.3 Gli strumenti Utili

Cominciamo a parlare di "**Software Utili**" andando ad analizzare quelle app o servizi che sono stati usati durante il lavoro per rendere più semplice la gestione o lo sviluppo del codice.

Partendo dal più comunemente utilizzato abbiamo **Git**, il quale fornisce un servizio di **controllo di versione distribuito** e **GitTea**, che è l'interfaccia web con cui ci collegiamo al repository. Utilizzando **GitHub Desktop** ovvero la GUI più utilizzata per la gestione delle cartelle di lavoro in remoto di Git, tutti i membri del team possono **agire simultaneamente** sul codice sorgente attraverso un sistema di **"pareggio"** dove ognuno ha sulla sua macchina personale una copia del codice originale alla quale può apportare modifiche, successivamente attraverso la **"commit"** esse verranno apportate al codice in maniera sicura evitando conflitti.. Le variazioni apportate da altri possono essere implementate sul codice locale di ognuno attraverso il metodo di **"pull"**, permettendo a tutti di avere sempre una copia **Up-to-Date** e **comune** dell'app. Git insieme a Jira, sono i due servizi utilizzati **presenti esclusivamente sulla intranet aziendale**.

Jira, software di **gestione delle segnalazioni**, il quale nasce come semplice strumento per la gestione di segnalazioni di Bug, risulta molto utile sotto molti aspetti dati per esempio la **divisione dei compiti nel gruppo**.

In un sistema ideale il membro di un gruppo di lavoro **"apre una Jira"** per segnalare la necessità di implementare una task che descrive all'interno della segnalazione, qua potrà specificare tipo, priorità

e componenti, per poi prendersi carico di tale compito, assegnarlo o attendere che qualcuno lo accetti. Tutto questo permette di avere una **divisione dei compiti precisa e stabile**, dove risulta chiaro lo scopo di una task e chi ci stia lavorando in quel momento.

Tenendo traccia delle persone, delle tempistiche e del tipo di task, Jira **fornisce uno storico preciso del lavoro svolto** ma anche del progresso dello sviluppo data l'attenzione anche alla versione del codice.

Vediamo poi **Jenkins**, strumento di supporto allo sviluppo software che fornisce dei servizi di **integrazione continua (CI)**, un sistema di “**controllo versione**” che punta a ridurre la mole di codice variato per aggiornamento, per impedire il cosiddetto “*integration hell*” ovvero la situazione in cui le variazioni apportate risultino eccessive e vadano conflitto con il codice precedente. Jenkins risulta simile a Git per la gestione delle cartelle del codice, ma principalmente è risultato utile perchè non si ferma al semplice upload, ma fornisce anche un **Server** su cui è possibile fare un **deploy accessibile da remoto**, questo rappresenta un sostanziale cambiamento in una fase iniziale di sviluppo rispetto a Tomcat o Wildfly data la possibilità di testare il codice “*comune*” caricato e in remoto e non solamente quello modificato sulla macchina.

4.4 Pattern e Codice

In questo capitolo affronteremo vari aspetti di “**semplificazione**” nell’ingegneria del software che sono risultati cruciali nello sviluppo durante lo stage.

Vedremo Pattern, Strumenti, Integrazioni e altre nozioni o standard utili al fine di migliorare il software, la sua manutenzione e la sua lettura.

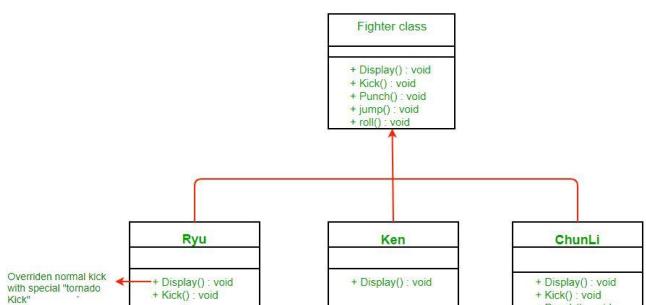
PATTERN

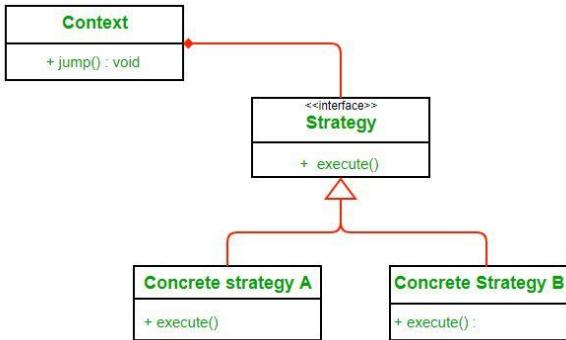
Un **Pattern** è una vera e propria “*soluzione progettuale generale ad un problema ricorrente*”, si tratta della descrizione di un **modello logico** applicabile al fine di risolvere un problema incontrabile in molteplici casi. Il loro utilizzo rappresenta spesso un buon modo per ridurre il **debito tecnico**, ovvero il pacchetto di difficoltà incontrabili durante le fasi di Progettazione e Sviluppo del codice (Costi, Esigenze di mercato, Mancanza di Documentazione o Collaborazione, [...]). Nel caso di Pattern orientati ad oggetti si evince una **definizione di relazioni e integrazione tra classi o oggetti**, mancando di individuare le classi applicative finali coinvolte, risiedendo quindi nel dominio dei moduli e delle interconnessioni. Nel caso di Pattern di livello più alto, ovvero **Pattern Architetturali**, invece si individuano vere e proprie strutture rispettate dall’intero sistema, la cui implementazione logica dà vita a un Framework.

STRATEGY PATTERN

Lo **Strategy Pattern**, è un **Design Pattern Comportamentale** che si basa sulla necessità di “*isolare un algoritmo*” per potervi accedere dove necessario dinamicamente a tempo di esecuzione.

Riportando questo esempio pensiamo ipoteticamente di star creando il gioco **Street**





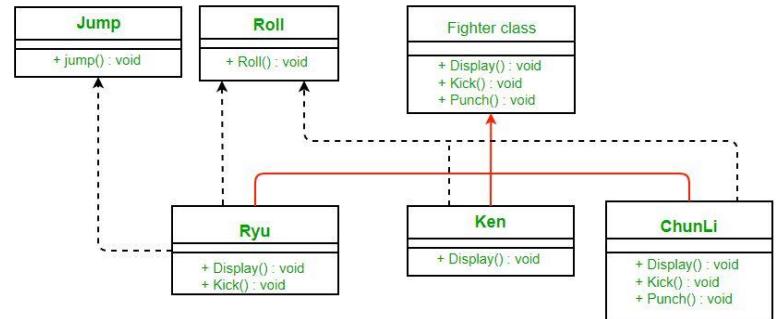
Fighter, dove abbiamo la **classe generale** Fighter, la quale viene **ereditata da tutti** i personaggi, permettendo comunque la sovrascrittura dei metodi della classe madre. Questa ereditarietà comporta che tutti i combattenti abbiano tutte le abilità, ma cosa succederebbe se uno di loro non potesse eseguire il metodo Jump()?

Un modo per risolvere il problema potrebbe essere un “Override” della classe inutilizzata per renderla vuota e impedirne l'utilizzo, ma questa ripetizione potremmo farla per decine se non centinaia di classi, o personaggi. Rendendo non solo la scrittura ma anche la manutenzione **più complicata**.

Una soluzione potrebbe essere **estrarre in delle interfacce le funzioni non comuni**, anche se comunque abbiamo il problema della ripetizione del codice data la mancanza di un'implementazione comune.

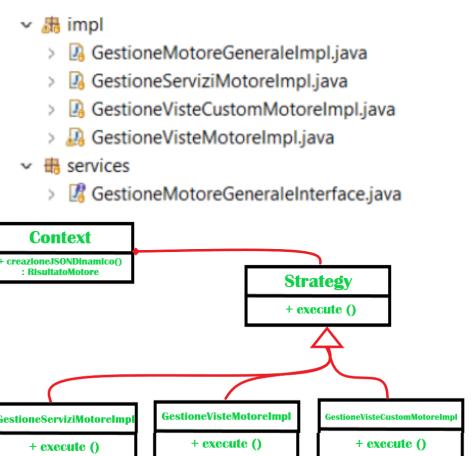
E' qua che lo strategy pattern diventa utile, dato il focus sulla composizione e non sull'ereditarietà, ci permette di abilitare dei metodi direttamente a runtime e renderli intercambiabili, creando solo le implementazioni varianti.

Ma vediamo come è stato utile durante il mio lavoro.



Seguendo la schematizzazione dell'esempio vediamo come all'interno del contesto sia presente il metodo “**creazioneJSONDinamico**” il quale restituisce un oggetto di tipo “**RisultatoMotore**” (Un oggetto composto dal JSON compilato e dal suo nome).

Data la natura comune del risultato, ma differente nella produzione, mi sono avvalso dello strategy pattern per permettere la **distinzione tra le classi**. Vediamo come sia importante distinguere **diversi metodi che portano allo stesso risultato**, specialmente in questo caso dove persino i dati utilizzati definiti nelle implementazioni siano completamente diversi. Un passo importante è stato quello dell'implementazione nel motore della **generazione di JSON Custom**, dato che possiamo immaginare l'utilità dello strategy pattern nella possibilità in cui i JSON custom possano diventare molteplici ma legati da un metodo comune.

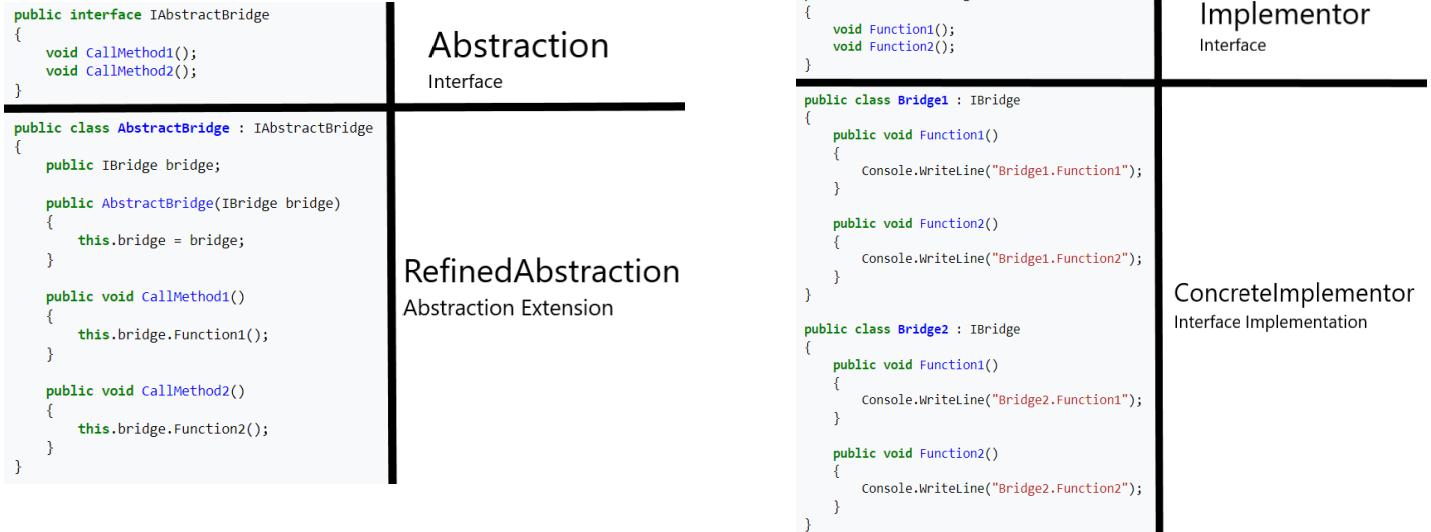
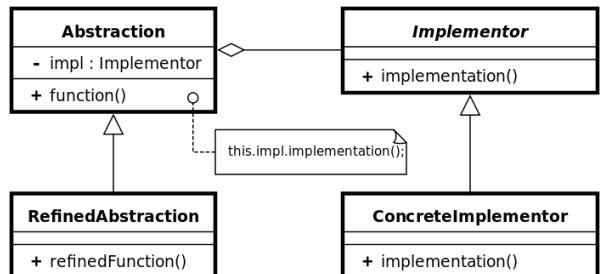


BRIDGE PATTERN

Il **Bridge Design Pattern** si basa sulla necessità di “*disaccoppiare un’astrazione dalla sua implementazione in modo che i due possano variare indipendentemente*”. Risulta importante nel caso di gerarchie di classi ortogonali e per favorire il binding, un meccanismo per cui il metodo che chiama un oggetto lo fa ricercandolo per nome a runtime.

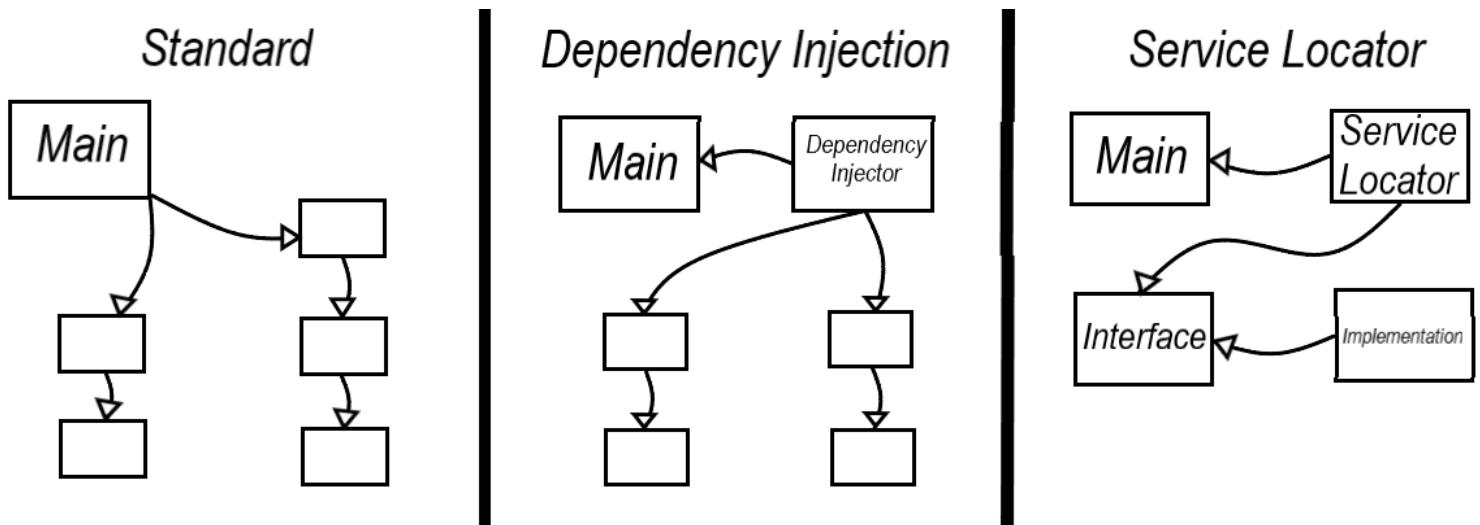
Questo schema mostra come l’”*Implementor*” ovvero l’interfaccia dell’implementazione, sia legata ad essa, “*ConcreteImplementor*”, solo attraverso una reference. Mentre la classe astratta “*Abstraction*” è la classe che definisce l’interfaccia senza però implementarla, passo che viene compiuto dalla “*RefinedAbstraction*”, ovvero l’estensione dell’astrazione.

Vediamo un esempio di Bridge Pattern scritto in C#.



SERVICE LOCATOR PATTERN

Il **Service Locator Pattern** è simile al Bridge Pattern nel suo tentativo di divisione tra astrazione ed interfaccia, con la differenza della presenza dello stesso Service Locator, ovvero la classe che funge da “*dispatcher*” in base alla richiesta. Questo sistema va a semplificare il sistema delle Dependency Injection, un sistema simile ma non sotto forma di listato ma di implementazione.



Abbiamo visto fino ad ora che generalmente “**dividere i lavori**” risulta essere sempre la scelta più precisa e utile, rendendo le cose facilmente leggibili e intercambiabili.

La funzione principale è quella di **run-time Linker**, per permettere a tutto il codice di avere la possibilità di accedere a componenti diversi nella loro istanza corrente. In casi di necessità e di ulteriore divisione possiamo immaginare un sistema di più Locator che gestiscono a un livello superiore la reindirizzazione dei metodi.

Vediamo ora come è stato utile durante il mio lavoro e anche come sono stati affrontati alcuni aspetti.

```
try {
    RisultatoMotore risultato = new RisultatoMotore();
    risultato = ServiceLocator.getInstance().getGestioneMotoreGenerale(path, estraiJsonBody(jsonBody));
    json = risultato.getJsonProdotto();

} catch (BusinessException e) {
    log.error(e);
    json = JsonEngineUtil.generaMessaggioErrore("Errore durante la ricerca");
}
return json;
}
```

Partiamo da vedere la sua chiamata, notiamo da `getInstance()` la presenza del concetto di Linker a run-time definita prima. Questo andrà a richiamare l’interfaccia del metodo specificato.

Nell’immagine successiva vediamo due metodi all’interno del Service Locator. E’ importante sapere che **nel Service Locator non dovrebbe comparire nessuna implementazione** in modo da mantenere il concetto di separazione tra layer, infatti le righe 43-45 rappresentano il modo corretto di scrittura del metodo, ovvero richiamare l’interface che restituisce l’istanza dell’implementazione dentro alla quale saranno i metodi a cui accedere. Differentemente le righe 47-65 rappresentano il sistema meno corretto ma che si è rivelato necessario per la gestione degli input, dato che qua verrà decisa dal codice il primo passo nella gerarchia di possibilità, ovvero se ricevere i dati da un DB o da un servizio REST.

```
43- public GestioneMotoreGeneraleInterface getGestioneMotoreGenerale() {
44-     return new GestioneMotoreGeneraleImpl();
45- }
46-
47+ public RisultatoMotore getGestioneMotoreGenerale(String path, Map<String, Object> valoriInput) throws BusinessException {
48-     log.debug("Service Locator - getGestioneMotoreGenerale");
49-
50-     DtoMotore input = new DtoMotore();
51-     input = GeneraDtoMotore().generaInput(valoriInput, path);
52-
53-     boolean isDB = TipoSorgenteEnum.DB.getCodice().equals(input.getVista().getSorgente());
54-     boolean isStandard = TipoPaginaEnum.STANDARD.getCodice().equals(input.getVista().getGradoComplessita());
55-
56-     if (isDB)
57-     {
58-         return new GestioneMotoreGeneraleImpl().creazioneJSONDinamico(input, isStandard);
59-     }
60-     else //isAPI = true
61-     {
62-         return new GestioneMotoreGeneraleImpl().creazioneJSONDinamico(input);
63-     }
64- }
65-
```

Nelle righe 59 e 63 vediamo anche lo Strategy Pattern spiegato precedentemente in azione nell’utilizzo dello stesso metodo “`creazioneJSONDinamico`” ma parametrizzato in maniera differente a seconda dell’input.

DTO PATTERN

Nell'immagine sopra relativa allo SP, alla riga 51 vediamo “**DtoMotore**”, oggetto detto **Data Transfer Object**, parte del **Pattern Strutturale DTO**.

Basati su un concetto di “**raggruppamento**”, questi oggetti consistono in un insieme di valori e vengono usati per evitare di spostare molteplici oggetti piccoli compiendo un numero maggiore di chiamate. E’ importante dire che **non presentano nessuna logica** ma soltanto la definizione degli attributi e dove sono necessarie le loro getter e setter.

Mantenendo l'esempio di DtoMotore, vediamo le sue variabili e cosa cambierebbe senza la sua presenza:

- private CollegamentoViste vista;
- private String template;
- private Map valoriInput;
- private List<String> listaPlaceholder;

Implementando questo sistema evitiamo di dover passare necessariamente 4 oggetti differenti per volta dove sono utili insieme.

DAO PATTERN

Spesso citato insieme al DTO abbiamo anche il **DAO**, **Data Access Object**, un altro **Pattern Strutturale** che ci permette di **isolare la logica dal layer di persistenza**, ovvero il database relazionale.

Il DAO nel nostro caso è composto dalla definizione di una variabile tipizzata secondo il POJO dentro cui i dati del database andranno ad essere salvati, e tutti i metodi che ci permettono di interagire con essi come **SELECT, UPDATE, DELETE, INSERT**.

Facendo così possiamo mantenere un rapporto con i database più chiaro e localizzato, facilitando l'utilizzo generale dei dati.

```
public class DtoMotore implements Serializable {  
    /**  
     * Type : VARCHAR Name : PATH  
     */  
    private String path;  
  
    /**  
     * Type : VARCHAR Name : NOME_VISTA  
     */  
    private String nomeVista;  
  
    /**  
     * Type : VARCHAR Name : GRADO_COMPLESSITA  
     */  
    private String gradoComplessita;  
  
    /**  
     * Type : VARCHAR Name : SORGENTE  
     */  
    private String sorgente;  
  
    /** Sets the value for path  
     */  
    public void setPath(String path){  
        this.path=path;  
    }  
  
    /** Gets the value for path  
     */  
    public String getPath(){  
        return path;  
    }  
  
    /*  
     *  
     *  
     */  
    private static final long serialVersionUID = -814466300106439315L;  
  
    private CollegamentoViste vista;  
    private String template;  
    private Map valoriInput;  
    private List<String> listaPlaceholder;  
  
    public List<String> getListaPlaceholder() {  
        return listaPlaceholder;  
    }  
    public void setListaPlaceholder(List<String> listaPlaceholder) {  
        this.listaPlaceholder = listaPlaceholder;  
    }  
    public String getTemplate() {  
        return template;  
    }  
    public void setTemplate(String template) {  
        this.template = template;  
    }  
  
    public CollegamentoViste getView() {  
        return vista;  
    }  
    public void setView(CollegamentoViste vista) {  
        this.vista = vista;  
    }  
    public Map getValoriInput() {  
        return valoriInput;  
    }  
    public void setValoriInput(Map valoriInput) {  
        this.valoriInput = valoriInput;  
    }  
}
```

DAO PATTERN

Spesso citato insieme al DTO abbiamo anche il **DAO**, **Data Access Object**, un altro **Pattern Strutturale** che ci permette di **isolare la logica dal layer di persistenza**, ovvero il database relazionale.

```
public class CollegamentoVisteDAO extends AbstractTableDAO {  
  
    private CollegamentoViste collegamentoviste;  
  
    public CollegamentoVisteDAO(CollegamentoViste collegamentoviste) {  
        super();  
        this.collegamentoviste = collegamentoviste;  
    }  
    protected String getSqlRetrieveObjectByKey(){  
        String sql = "SELECT * from COLLEGAMENTO_VISTE where PATH=?";  
        return sql;  
    }  
  
    protected void setStatementRetrieveObjectByKey(PreparedStatement st) throws SQLException{  
        int index=1;  
        st.setString (index++ , collegamentoviste.getPath());  
    }  
  
    protected String getSqlRetrieveObjectByWhere(){  
        String sql = "SELECT * from COLLEGAMENTO_VISTE where 1=1 ";  
        if(collegamentoviste.getPath()!=null)  
            sql += " and PATH = ?";  
        return sql;  
    }  
}
```

POJO

```
public class CollegamentoViste implements Serializable {  
    private static final long serialVersionUID = 1L;  
    public CollegamentoViste() {  
        super();  
    }  
    /**  
     * Type : VARCHAR Name : PATH  
     */  
    private String path;  
  
    /**  
     * Type : VARCHAR Name : NOME_VISTA  
     */  
    private String nomeVista;  
  
    /**  
     * Type : VARCHAR Name : GRADO_COMPLESSITA  
     */  
    private String gradoComplessita;  
  
    /**  
     * Type : VARCHAR Name : SORGENTE  
     */  
    private String sorgente;  
  
    /** Sets the value for path  
     */  
    public void setPath(String path){  
        this.path=path;  
    }  
  
    /** Gets the value for path  
     */  
    public String getPath(){  
        return path;  
    }  
}
```

Per **POJO, Plain Old Java Object**, si intende un **oggetto ordinario** non legato a nessuna restrizione particolare, utilizzato solamente per la gestione di se stesso evitando l'utilizzo di logica.

Come vediamo dall'esempio è stato usato mantenendo il **legame con il DAO** assegnandogli una variabile per ogni colonna della tabella di database. Vediamo anche che sono presenti solo variabili tipizzate e non oggetti, e le loro rispettive Getter e Setter.

MODELLI, STANDARD e altro

Allontanandoci dal concetto di Pattern vediamo ora tutti quegli aspetti che hanno contribuito a semplificare delle particolarità del lavoro.

• REST - Representational state transfer

E' uno **stile architetturale** per sistemi distribuiti, non prevede un concetto di sessione essendo **stateless**, ed è stato utilizzato come **sistema di trasmissione dati**.

Il funzionamento di questa architettura si basa su HTTP e prevede una struttura ben definita di URL per identificare univocamente le risorse, con cui si può agire utilizzando i metodi HTTP specifici (vedi 4.1 Insomnia).

I **Servizi o Risorse Rest** sono state un punto importante per la loro immediatezza nel passaggio di informazioni. Grazie ad essi possiamo far comunicare backend e frontend con semplicità, passando informazioni di vario tipo, in questo caso JSON e FILE.

```
@PatchMapping("/inserisciRispostaConsulito/{idConsulito}")
ResponseEntity<Object> inserisciRispostaConsulito(HttpServletRequest request,
    @PathVariable("idConsulito") String idConsulito) {
    /*
     * log.info("ID Consulito che voglio inserire la risposta: " + idConsulito);
     */
    ObjectMapper mapper = new ObjectMapper();
    String input = new String();
    try {
        input = mapper.readValue(request.getReader(), String.class);
    } catch (IOException e) {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
    log.info("Input inserito: " + input);
    /*
     */

    UploadComponent.tsx
    const handleSubmit = async (event:any) => {
        event.preventDefault()
        const formData = new FormData();
        formData.append("file", selectedFile);
        try {
            uploadService.setFileUpload(formData);
        } catch(error) {
            console.log(error)
        }
    }

    UploadService.tsx
    class uploadService {
        setFileUpload(file: any){
            console.log(file);
            return http.post(
                "/uploadFile",
                file,
                [headers: { "Content-Type": "multipart/form-data" }]
            );
        }
    }
    export default new uploadService();
```

Tramite l'utilizzo della **Comunicazione a Endpoint** il passaggio delle variabili è semplificato, sia per esempio nel recupero di valori dal database da mostrare nel frontend, che alla loro modifica una volta inseriti nuovi campi e rimandati aggiornati al BE.

In questo esempio di Endpoint nel BE in Spring Boot vediamo l'utilizzo dei **@metodi HTTP**, insieme alla definizione di un **/URL** specifico con anche l'indicazione di una **{variabile}** e l'utilizzo degli **Stati HTTP** come per esempio il “500”, Internal_Server_Error.

Vediamo poi nel Typescript facente parte del FE come sia semplice il passaggio anche di FILE veri e propri grazie, all'utilizzo di chiamate HTTP, dove diamo semplicemente:

- l' **/URL**
- **FILE**, in questo caso utilizzando la libreria “*axios*” possiamo generare un oggetto con “nome” e l'insieme delle specifiche dei file
 - gli **{HEADERS}**, contenenti informazioni aggiuntive sulla risorsa, in questo caso il suo tipo **“multipart/form-data”**.

• ATTENZIONE AL CODICE

In molte situazioni possiamo avere una necessità di un codice più veloce e più sicuro, non solo dipendente dal ciclo, algoritmo, o ripetizione generale di codice, ma anche dalla tipizzazione delle variabili.

Durante il lavoro in EDAE ho approfondito la differenza tra il tipo **“StringBuffer”** e **“String”**.

- Il tipo SB ha una **lunghezza variabile** al contrario di quella del tipo String.
- E' **più veloce da elaborare** per il calcolatore, migliorando le prestazioni del codice.

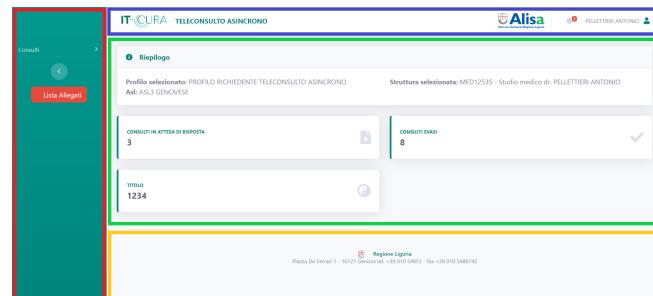
- E' **mutable**, ovvero riassegnabile, al contrario del tipo String che al posto che venir riassegnato crea una copia di se stesso
- E' **più leggero** e occupa meno memoria.

Legato alle differenze del codice vediamo anche concetti di base come l'utilizzo di **ClassLoader.getResourceAsStream(name)** per nascondere il path di una risorsa, invece di new File per esempio che necessita di dover scrivere il path completo nel codice per la retrieve.

• RICICLO DEL CODICE

Specialmente nel caso di Liguria Digitale sarà normale e probabile trovare casi di necessità simili, se non uguali, come per esempio:

- Il Layout per certi prodotti web è stato reso standard e comprende **TOPBAR**, **SIDEAR**, **BODY**, **FOOTER**. Varieranno poi le icone, le funzioni, il contenuto, ma la struttura resta la stessa.
- Un altro comportamento che ho reputato utile, specialmente nei casi dove è normale riutilizzare più volte le stesse strutture, è la creazione di una **SKELETON APP**, ovvero la standardizzazione di un modello attraverso la rimozione dei contenuti specifici e non necessari al riutilizzo. Per quanto sia una pratica utile, può risultare inusuale o inutile nei casi in cui la necessità di portabilità sia scarsa.



• VERSIONAMENTO e BRANCH

- Un aspetto importante per la semplificazione del lavoro di gruppo è il corretto versionamento di un app in sviluppo. La buona norma recita che **è importante non creare numeri eccessivi di branch** (*branch, o "ramo", è un'estensione di un'applicazione che parte dal codice sorgente dell'applicazione stessa*) con lo scopo di concentrarsi sullo sviluppo progressivo della versione in sviluppo, mentre vengono corretti i bug emersi dalla versione di test. Mantenendo un progresso "+.1" (esempio: la versione 1.0.x in test e 1.1.x in sviluppo) possiamo preoccuparci di fare soltanto una **merge** (*merge è il gesto di unire in un unico codice sorgente le modifiche apportate su un branch differente*) tra le correzioni del test e il progresso di sviluppo, al contrario se per ogni feature facessimo un branch differente, in breve tempo saremmo costretti a fare la merge di più versioni, aumentando il rischio di conflitti tra codici modificati in maniera differente.

• LA GESTIONE DEI SERVER

- Dopo aver visitato la zona dell'azienda utilizzata dai sistemisti sono emersi dei dati interessanti dal lato di **ottimizzazione** riguardo la gestione dei server.
Vediamo come la **virtualizzazione delle macchine** si riveli importante sotto molti aspetti:
 - Partiamo per esempio dal fatto che su 8 macchine fisiche troviamo installate 558 macchine virtuali, con un rapporto medio di **1 reale : ~69 virtuali**
 - Viene tenuta molta attenzione alla **partizione e ai metodi RAID** per una fruizione di servizi continuativi, vediamo insieme il concetto di **Disaster Recovery e Business Continuity**, entrambi consistono in un insieme di misure tecniche / logistiche / organizzative, per affrontare al meglio l'evenienza di un guasto o malfunzionamento di una o più macchine fisiche, con la

differenza che il Disaster Recovery punta a non perdere dati e funzioni nell'evenienza di un'emergenza, mentre la Business Continuity punta a mantenere il servizio in corso attraverso lo spostamento di esso in tempo reale tra la macchina in emergenza e una pronta all'uso.

- Con la virtualizzazione abbiamo un **risparmio notevole di energia** dato che per quanto le macchine usate consumino di più essendo loro spinte a prestazioni molto alte, rimangono di gran lunga sotto le soglie che raggiungerebbero lo stesso numero di macchine non virtualizzate.
- Come per l'energia abbiamo ovviamente un **risparmio di spazio e minor necessità di manutenzione**
- Vi è inoltre la possibilità di una **migliore gestione delle prestazioni**, andando, nel caso di necessità di maggiore potenza, a spostare il file che rappresenta la macchina virtuale su un server libero più prestante.

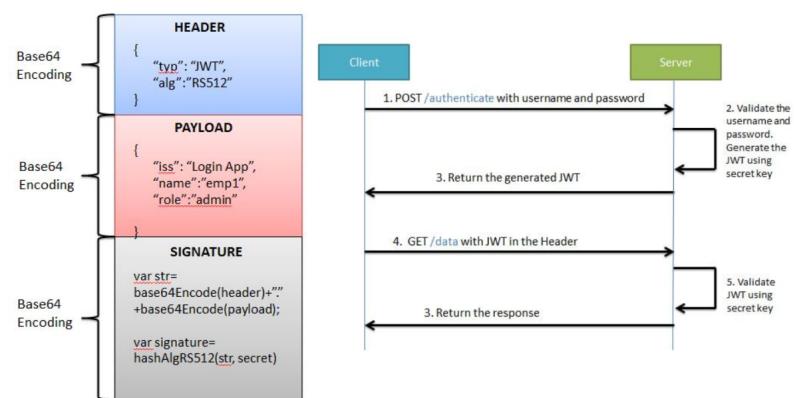
Grazie a questi accorgimenti abbiamo la possibilità di concentrarci sull'incrementare la potenza fisica di poche macchine per sostenere le varie virtualizzazioni.

Passando alla configurazione software delle macchine vediamo l'utilizzo di **Ansible**, un software di **automazione per la gestione dei processi di configurazione** di macchine linux-like e windows, che viene utilizzato per implementare la metodologia "**infrastructure as code**". L'utilizzo dei file **.yml** con Ansible ha **drasticamente ridotto le tempistiche** di creazione e configurazione (7-10 giorni di processo ridotti a **qualche ora!**), specialmente se pensiamo agli enormi numeri di macchine mantenute da Liguria Digitale. Questa automatizzazione consiste nello scrivere un file che contiene tutte le caratteristiche da assegnare al sistema, facendo così si *riduce la possibilità di errore* nell'inserimento delle variabili. Un importante aiuto fornito da questo sistema è dato in ambito di **manutenzione**, per esempio durante i test di compliance quando un software viene caricato sul sistema e bisogna controllare che le specifiche siano esatte per evitare errori. Anche nel caso in cui un software vada spostato per motivi di prestazioni, possiamo assegnare a un "gruppo" di macchine, la stessa configurazione per darci la sicurezza di successo in caso di spostamento dei file internamente al gruppo (anche nel caso di uno spostamento tra ambienti **STR** sviluppo, test, rilascio). L'unica difficoltà nell'utilizzo di un software di automazione simile consiste nel linguaggio specifico, cosa che troviamo anche in software competitor come **Terraform**.



• SICUREZZA DI BASE

- Vediamo un concetto di cyber security di base, ovvero l'utilizzo di **JWT**. Il **JSON Web Token** è uno standard che definisce una misura di sicurezza contenuta e sicura per la trasmissione di informazione sotto forma di oggetto JSON. La **firma digitale** del JWT garantisce la **sicurezza nella trasmissione**

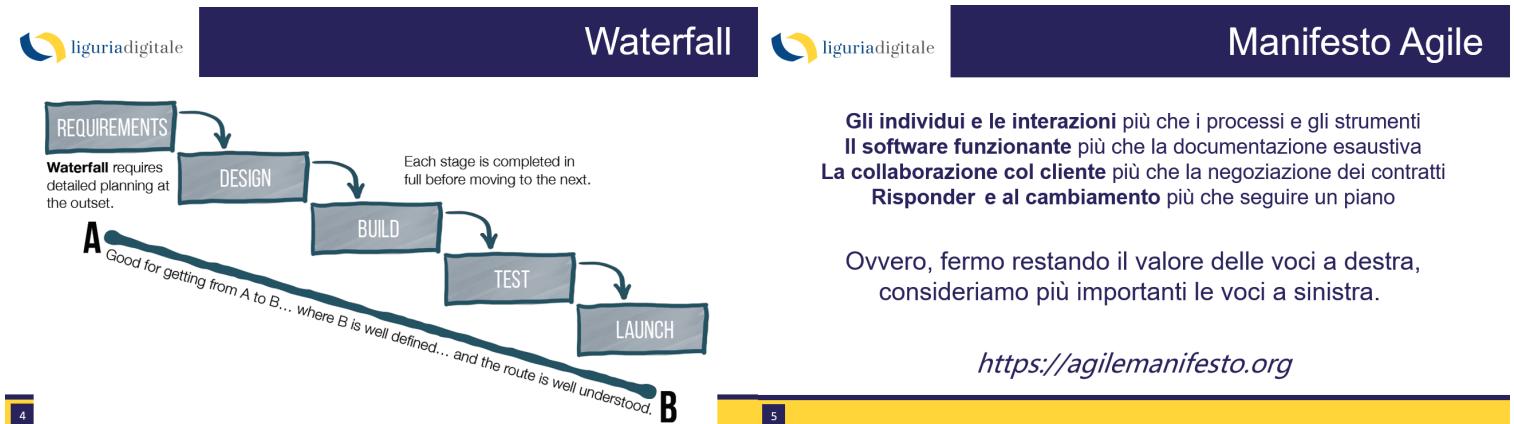


quando viene fornito a seguito della prima autenticazione del client. La sua struttura del

JSON è divisa in tre parti, { **HEADER**}, che contiene il typ di valore JWT e l'**algoritmo di hash** (algoritmo di Crittografia), il **PAYOUT**, ovvero i dati che stiamo cercando di passare, notiamo come comunque come questi restino visibili, rendendo rischioso passare informazioni sensibili utilizzando solo il JWT, e la **FIRMA** ovvero una variabile composta da header e payload crittografati con l'algoritmo di hash e un valore segreto concesso dal server }.

- METODOLOGIE

Agile_Workflow.pptx - Matteo Scaldaferrri



- Per **metodologia a cascata**, o **Waterfall**, si indica quell'approccio allo sviluppo software che enfatizza una progressione lineare dall'analisi fino alla manutenzione. Questo modo di procedere comporta una **unidirezionalità** nelle azioni: analisi → design → implementazione → test → rilascio, lasciando però poco spazio alla rielaborazione nel caso di un imprevisto, o di una modifica. Per quanto questo metodo sia il più diretto, data l'enorme possibilità di variazioni, rischia di rappresentare più un problema che una soluzione. A contrapporsi a questo sistema "diretto" vediamo la metodologia **Agile**, la quale presenta un sistema più "**a step**" non focalizzato sulla consegna diretta del prodotto finito, ma sul mostrare il lavoro fatto **periodicamente** per procedere in base al feedback, sia del cliente che del gruppo di lavoro, il quale resta costantemente aggiornato sul lavoro svolto e da svolgersi. Un altro aspetto importante è appunto la gestione dei team di sviluppo, dato che secondo questa metodologia è importante mantenere **gruppi piccoli, polifunzionali**, che possono **autogestirsi** e variare la loro analisi in base alle richieste variabili del cliente. Come sotto-metodologie possiamo individuare **Scrum** e **Kanban**, le quali si distinguono dal loro diverso utilizzo del **Backlog** di prodotto e della gestione delle tempistiche. Ma cosa è il Backlog? E' una **lista** di funzioni, feature, metodi, correzioni, fatti sulla base dei feedback dati dal cliente e dagli stakeholder analizzati dal team di sviluppo. I **team Scrum** lavorano per **Sprint**, ovvero periodi di due settimane dove all'inizio durante il meeting si scelgono e assegnano agli sviluppatori compiti prioritari che prevedono di poter finire in uno Sprint. Tutti i compiti terminati vengono rilasciati in attesa del feedback del cliente, mentre quelli non terminati potranno essere riassegnati allo Sprint successivo. E' importante notare l'importanza dei meeting conclusivi di **Sprint Review** per tenere conto di cosa è stato fatto, cosa si poteva fare meglio e cosa si farà. Un **team Kanban** al contrario non segue lo stesso utilizzo dei Backlog dato che nel momento in cui una mansione viene dichiarata come "**to do**" (da fare) questa può essere già presa in carico. Mentre i compiti assegnati durante uno Sprint sono prioritari, Kanban non utilizzando

periodi di tempo prestabiliti resta un processo **continuo**, dove si dà più importanza alla mole di lavoro rispetto che al tempo; dichiarando quanti compiti possono essere assegnati, uno sviluppatore avrà sempre un numero di “**slot di produzione**” che nel momento in cui sono liberi saranno utilizzati per l’assegnazione di un compito.

- **L'UTILITA' DELLA PERSISTENZA**

- Per **Persistenza** in informatica intendiamo la capacità di un dato di *sopravvivere indipendentemente dal runtime del codice*. Molto spesso infatti parliamo di Database, o di storage periferici, legando l'utilizzo di essi a file, o tabelle di dati, ma durante lo stage ho individuato un aspetto più **gestionale**. La persistenza in EDAE veniva usata per la definizione della struttura di ricerca dei vari dati o template.

COLLEGAMENTO_VISTA (PATH) → **STRUTTURE_VISTA** (NOME_VISTA) → VISTE:**NOME_VISTA**

Questo aspetto è importante poiché rende il codice **più leggero** dato che, seguendo il sistema di EDAE, una soluzione lato codice potrebbe essere l'utilizzo di una **Mappa Multidimensionale** (equivalente in type:Map del Jagged Array) contenente la gerarchia ad albero delle viste e dei dati, dando lo **stesso livello di collegamento** ma andando a **influire sulle prestazioni** del codice data la costante necessità di ciclare la Map per la ricerca della posizione o dei dati.

- **LOGGING**

- “**Loggare**” rappresenta l’azione di scrivere qualcosa su un file di log, che può essere un info di posizione nel codice, un warning per un valore mancante, un errore specifico o altro. L’utilizzo del file di log come **record** è un processo fondamentale per *tenere traccia delle azioni e degli eventi del codice* messi in ordine di successione temporale.

Un esempio

Nel codice

```
public class NomeClasse {  
    private static Log log = LogFactory.getLog(NomeClasse.class);  
    log.info("Testo del log");  
}
```

Nel file di log

aaaa-mm-gg hh:mm:ss,ms **INFO** [MioProgetto.NomeClasse] **Testo del log**

Esistono varie librerie che ci permettono di loggare, alcune necessitano del path del file su cui scrivono, alcune lo fanno sul terminale dell’IDE, altri sui file del server, e nel mio caso ne ho visti principalmente due tipi, **LogFactory** e **Slf4j**.

La più particolare è Slf4j, implementata nel mio caso grazie a una **annotazione** di SpringBoot appunto, **@Slf4j**, che ci permette di utilizzare un campo log preconfigurato.

L’azione di log risulta particolarmente utile per una questione di **problem solving** grazie alla visualizzazione dello storico degli eventi e delle azioni su cui possiamo individuare l’errore o la mancanza.

4.5 La grafica e le sue regole

Passiamo ora all'individuazione di standard e regolamenti visti nel settore della grafica web. Partendo dal lato sviluppatore vediamo la presenza di una libreria open source chiamata **Bootstrap Italia**, fornita dal **Dipartimento per la Trasformazione Digitale** in collaborazione con l'**AGID, Agenzia per l'Italia Digitale**.

“ La libreria Bootstrap Italia è il modo più semplice e sicuro per costruire interfacce web moderne, inclusive e semplici da mantenere. “



All'interno del loro sito vediamo tutta la documentazione, dalle API della Pubblica Amministrazione che usano BI, alla lista di leggi e norme che sono rispettate durante lo sviluppo fino a modelli predefiniti per i Designer.

Un altro aspetto importante dello sviluppo web è l'**accessibilità**, la quale viene controllata al termine di un progetto grazie a strumenti di valutazione grafica.

WAVE ci permette di individuare in una qualsiasi pagina web tutti quegli elementi che:

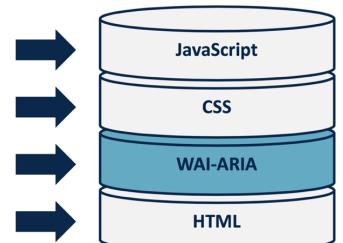
- possono portare a **confusioni**, come per esempio buttoni mancanti di descrizioni
- potrebbero essere **migliorati per essere più esplicativi** come la necessità di aggiungere un placeholder all'interno di un campo di inserimento testo.

- Che risultano un **problema** per una questione di disabilità visive come due colori facilmente confondibili o con poco contrasto tra testo e background
- Che compongono la pagina e la loro **disposizione strutturale**

WAVE dà anche la possibilità di analizzare anche gli elementi **ARIA** di una page, ovvero **Accessible Rich Internet Applications** e rappresentano una serie di attributi per migliorare l'accessibilità dei siti web per le disabilità e l'utilizzo delle **Tecnologie Assistive**.

Le funzioni ARIA variano per utilizzo da semplici regole di codice come il corretto utilizzo dei tag HTML

Think about accessibility at every layer of the front-end development stack



`<button type="submit" onClick="submitForm();>INVIA</button>` Corretto
`INVIA` Errato

ma anche l'utilizzo di etichette descrittive le quali possono essere lette per utenti con disabilità visive, rendendo la pagina più accessibile

`<a aria-label="Read More about Liguria Digitale" href="/path/to/your/page">Leggi di più`

CAPITOLO 5 - LO SVOLGIMENTO DELL'ATTIVITÀ

5.1 Studio ed Apprendimento di nuovi Strumenti

Scaldaferri - Di Dio

```

riscrivere / estensione
BE: https://gitlab.java.it/liguriadigitale/.../gio/motore/CollegamentoViste.java
Get Dinamica & Check code Riccardo creazione della tabella
La repo di Git
"un servizio" -- vista - oggetto -- MOTORE
INPUT -- riceve il nome della risorsa dove stanno i dati (chiamato ora "path")
dove si trova la risorsa (ora è una vista)
ricava il template (con i segnaposto da sostituire)
ricava la struttura dell'oggetto che riceverà
ricava l'elenco dei parametri necessari per chiamare la risorsa
2. Fatto questo il motore chiama la risorsa (ora è una vista, usa RicercalIntelligenteDAO, dovrà essere una API),
e riceve lista di oggetti
3. Effettua la sostituzione nel template con i dati ricevuti dalla risorsa.
  
```

GestioneMotoreImpl

```

CollegamentoViste getView(String path)
dato il path restituisce l'oggetto CollegamentoViste relativo
List<StrutturaVistaCompleto> getStrutturaVista(String vista)
data la vista restituisce l'oggetto List<StrutturaVistaCompleto> relativo
List<ParametrInput> getParametrInput(String vista)
data la vista restituisce l'oggetto List<ParametrInput> relativo
String getJson(String path, Map<String, Input>)
dann il path e la mappa <chiave valore valoriInput
  
```

```

$ cd /opt/liguriadigitale/gio/motore
$ java -jar target/Motore-0.1.jar
...
  
```



L'apprendimento progressivo dei nuovi strumenti è stato svolto direttamente sul campo grazie al sostegno dei colleghi / tutor **Matteo Scaldaferri e Riccardo Riggi**, i quali mi hanno introdotto all'ambiente lasciando poi a me lo spazio per imparare da solo.

Utilizzando ambienti di test ho avuto sempre la possibilità di fare passi avanti grazie a un sistema **trial-and-error** preceduto da uno studio di utilizzo e funzionalità.

Un punto importante emerso nel corso dello stage è la **necessità di versatilità** la quale è fondamentale in un ambiente dove bisogna esser pronti ad affrontare problemi differenti tra loro.

Lo sviluppo di conoscenze nell'ambiente aziendale è stato spinto anche dalla partecipazione alle **riunioni** con i team, cominciando ad approcciarmi con vari settori di lavoro e le loro diversità, sempre prendendo appunti i quali poi sono stati utili nell'apprendimento e nella stesura della tesi. Dagli incontri con i sistemisti o ingegneri del software, al semplice confrontarsi con i colleghi del team si può sempre scoprire una nuova soluzione a un problema magari già superato.

5.2 Modello Realizzativo

Vediamo adesso i processi di analisi che sono emersi durante il lavoro, e le loro possibili soluzioni.

Vorrei inoltre approfondire la scelta del tema della semplificazione. A seguito della conclusione di EDAE ho deciso di non eliminare definitivamente dal Project Work uno dei due lavori svolti, motivo per cui ho cominciato ad analizzare aspetti condivisi da entrambi, qua sono emersi Pattern e Standard, sia nel codice che nella metodologia del lavoro. Accumulando tutti i punti ho cominciato a vedere il tema della semplificazione riportato in vari modi e settori da cui la decisione di renderlo il fulcro della tesi.

Durante la realizzazione di EDAE ho lavorato utilizzando un **ambiente di test** usufruendo di tabelle di database di prova con dati statici, facendo così potevo inserire i link degli endpoint di restituzione dei dati dei JSON. Per i dati da inserire ho deciso di utilizzare inizialmente il sito npoint.io per una questione di comodità, dato che mi permetteva di creare un endpoint che restituisse il JSON inserito chiamando il link fornito. Dato che il sito aveva un traffico di dati **decisamente intrusivo**, capitava di avere rallentamenti di vari secondi nel recupero dei dati, spingendomi a dover cercare una soluzione, trovata nell'utilizzo di [Node.js](#). Con Node ho creato i miei endpoint di test personali in JavaScript creando un progetto in BE che utilizzava una porta del localhost con vari URL mirati per il recupero dei Template e dei dati `localhost:8080/{path}+“Template” / localhost:8080/{path}`.

Per il Teleconsulto è emerso un problema legato alla collaborazione di vari team, cioè la **dipendenza da gruppi esterni** per il proseguimento del lavoro. Quando si parla di lavoro di gruppo si pensa spesso al gruppo di persone impegnate in una mansione pari alla propria, ma in questi casi diversi gruppi diventano dipendenti l'uno dall'altro, rallentandosi nel caso manchi la “sintonia”.

5.3 Principali Risultati Raggiunti e Output



Elastic Data Access Engine è stato **concluso** e ultimato con la merge del feature branch sviluppato, insieme al source code originale, portando a un EDAE v2.

Partendo dal concetto di riutilizzo per cui è stato creato, l'engine si trova a poter essere richiamato da una qualsiasi web app la quale abbia la necessità di ricevere un JSON compilato.

Andando a vedere come varia nella sostanza, il progetto implementa una persistenza differente all'originale, la quale interagisce pienamente con la versione scritta.

Mantenendo EDAE v1 per il suo utilizzo delle pagine custom, per i metodi e DAO introdotti inizialmente per essere localizzati, abbiamo un quadro completo del lavoro svolto, che con la successiva modifica del Service Caller e Locator, la gestione delle API, e i Pattern introdotti, si modifica per essere più veloce e più portabile.



Teleconsulto e app cure domiciliari

Dopo vari cambiamenti l'app delle cure domiciliari al termine dello stage è arrivata alle porte del processo di rilascio come versione 1.0.0.

Mentre il Teleconsulto si prepara agli ultimi ritocchi per un possibile rilascio tra Dicembre 2022 e Gennaio 2023.

	Id	Medico Richiedente	Data richiesta	Assistito	Data risposta	
	1	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>
	2	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>
	3	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>
	4	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>
	5	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>
	6	COGNOME NOME	17/10/2022	COGNOME NOME	DATO VARIABILE	<button>Visualizza</button>

CAPITOLO 6 - CONSIDERAZIONI FINALI

6.1 Crescita Personale e Consolidamento Tecnico

Tecniche

- React
- Wildfly, Tomcat e le loro differenze
- Servizi REST
- WAVE
- Spring Boot
- Design Pattern
- L'importanza del Clean Code
- Metodologie
- L'importanza del versionamento

Personal

- La **necessità** di:
 - Versatilità
 - Problem Solving
 - Pensiero computazionale
- Il rispetto delle tempistiche
- Le proprie tempistiche di sviluppo
- La divisione dei compiti

Le competenze tecniche sono state importanti per poter aggiungere al mio portfolio delle nuove conoscenze, come per esempio **React**, una delle librerie più utilizzate internazionalmente al giorno d'oggi. Aspetti come **Clean Code** e **Versionamento** sono spesso tralasciate durante l'insegnamento che lavorando direttamente in azienda si sono rivelate invece molto importanti. Termino il mio periodo di stage anche con una **maggior considerazione della persistenza**, non solo come storage di dati ma anche come sostegno all'organizzazione di gerarchie di progetto più ampie.

Per la parte di crescita personale ho appurato che le **soft skills** spesso risultino **più necessarie delle hard skills** per potersi muovere al meglio in un ambiente. Specialmente in quanto figura junior è più probabile che i colleghi si aspettino che io impari in fretta piuttosto che conosca già gli argomenti, questo rende obbligatorio uno sviluppo e un apprendimento più rapido. La divisione dei compiti nonostante sia una via più semplice e veloce porta con sé molti ostacoli come la necessità di attendere che certe mansioni siano terminate per poter continuare e questo può portare a ritardi o attese dovute da diversi motivi. In un ambiente dove la versatilità è necessaria risulta fondamentale prendere molte pagine di **appunti** anche solo su come risolvere problemi minimi, dato che spesso i problemi banali possono essere quelli più difficili da individuare.

Fonti:

intranet.liguriadigitale.it
wikipedia.org
mybatis.org
redhat.com
geeksforgeeks.org
baeldung.com
tutorialspoint.com
italiancoders.it
npoint.io
hpe.com
i colleghi