Paolino Angeletti

Senior Back-End Engineer with Front-End Development Experience (Junior Level)

Informazioni personali

Data di nascita	04/05/1993
Nazionalità	Italiano
Livello inglese	Livello B1

Contatti

Email	paolinoangeletti@gmail.com
Cellulare	+39 320.29.50.351
Linkedin	linkedin.com/in/paolino-angeletti-64484211b
Indirizzo	via Trivice d'Ossa, 80030, Cimitile, Napoli

Istruzione

Laurea Triennale in Informatica
Università Federico II, Napoli • 2012–2016 • Votazione: 100/110

Diploma Perito Commerciale e Programmatore
ITCG Masullo-Theti, Nola (NA) • 2009–2012 • Votazione: 74/100

Esperienze professionali

Teoresi Group S.p.A. (Consultant @ Hitachi Rail)
Napoli • Giugno 2024 – Presente

- Progetti: 360Motion, 360FlowVehicle (ambito Railway)
- Sviluppo microservizi in Java Spring Boot e Python

- Utilizzo avanzato di: Docker, Kafka, Keycloak, Redis, PostgreSQL, AKHQ, JMeter, SonarQube, Test Containers
- Progettazione e refactoring SDK e librerie interne
- Gestione test di integrazione tramite libreria proprietaria Test Core
- Protocolli: HTTP, MQTT
- Sicurezza: Crittografia, Keycloak, JWT

Advise S.R.L.
Mugnano del Cardinale (AV) • Giugno 2016 – Gennaio 2024

- Progetti Android (Java) & Back end (Node.js, Python)
- Project Manager su vari gestionali (NegozioPerfetto, MBE LAB, MagazzinoPerfetto)
- Integrazione con AWS, RFID, Blockchain, WooCommerce
- Realizzazione APP offline-first sincronizzate con MySQL/DynamoDB
- API REST, notifiche push (Firebase), stampa Bluetooth (ZPL)

Software Business S.R.L.
Mugnano del Cardinale (AV) • Tirocinio universitario • Marzo 2016 – Agosto 2016

- Progettazione e rilascio piattaforma ordini B2B integrata SAP

Competenze tecniche

Linguaggi: BeanShell, CSS, HTML, Java, JavaScript, Node.js, Python, Swift, VB.NET

Framework / Tooling: Android, Django, Flask, Git, Git Flow, Gradle, Lombok, Maven, Retrofit, SonarQube, Spring Boot

Cloud Services: AWS (Lambda, API Gateway, DynamoDB, Aurora, Cognito, S3, Batch, ECR, EC2, CloudWatch), Azure DevOps

Container & Orchestration: Docker, Docker Compose, Kubernetes (base)

Database:

- **Relazionali**: PostgreSQL, MySQL, SQL Server
- **NoSQL**: DynamoDB, Redis (con ReJSON)

Message Brokers: Kafka (AKHQ), MQTT

Testing & CI/CD: Azure DevOps Pipelines, GitHub, Jasmine, JUnit, JMeter, Swagger, Test Containers

Pattern: Singleton, Factory, Mediator, Observer

Documentazione & PM: Doors, Microsoft Planner, Project Place, Confluence, JIRA

Hardware & IoT Integration: RFID, ZPL

Approfondimenti tecnici

Le sezioni seguenti includono una descrizione tecnica estesa delle esperienze lavorative, delle tecnologie utilizzate e delle architetture implementate. Sono pensate per chi desidera un approfondimento tecnico sui progetti entreprise seguiti, le soluzioni adottate e il contributo personale.

1 Esperienze professionali

Di seguito, l'elenco delle mie esperienze professionali fino a questo momento in ordine dalla più vecchia alla più recente.

1.1. Software Business S.R.L.

Data inizio	Marzo 2016
Data fine	Agosto 2016
Luogo	Mugnano del Cardinale (AV)
Sito web	www.softwarebusiness.it

Questa esperienza è nata come opportunità di tirocinio messa a disposizione dall'università. Il mio obiettivo finale era il rilascio di una piattaforma web user-friendly avente lo scopo di creare ordini clienti in **SAP.** Tale applicativo doveva essere utilizzato dagli stessi customer finali, che quindi potevano ordinare merce autonomamente, stile e-commerce.

1.2. Advise S.R.L.

Data inizio	Giugno 2016
Data fine	Gennaio 2024
Luogo	Mugnano del Cardinale (AV)
Sito web	www.advise.it

La mia esperienza in questa azienda si può suddividere in due fasi:

All'inizio, le prime attività riguardavano la manutenzione degli applicativi già in produzione, scritti prevalentemente con tecnologia .NET e database SQL o MySQL.

In questa fase mi sono occupato sia dello sviluppo Back-End che Front-End, con l'ausilio delle classiche tecnologie HTML, CSS e Javascript. Nessun framework in particolare è stato utilizzato.

Dopo circa un anno dalla mia assunzione, sono stati avviati alcuni progetti per lo sviluppo di APP, nello specifico Android e Gestionali di magazzino (ambito logistico), di cui sono diventato, col tempo, Project Manager. I progetti vengono descritti più ampiamente nella sezione dedicata.

Queste APP sono state sviluppate, quindi, con l'utilizzo del framework Android e l'integrazione di alcuni servizi AWS.

1.3. Teoresi Group S.P.A.

Data inizio	Gennaio 2024
Data fine	Esperienza attuale
Luogo	Napoli, Via Ferrante Imparato
Sito web	www.teoresigroup.com

Teoresi è una multi-nazionale in ambito di consulenza ingegneristica. Come tale, sono stato assegnato, nel tempo, a diversi progetti per diversi clienti finali, di cui parlo nei paragrafi successivi.

In questo periodo ho vissuto politiche di smart-working ibrido, alternando così giornate in presenza (sia in sede Teoresi che sede cliente) con giornate di smart.

1.3.1. Hitachi Rail STS S.P.A.

Data inizio	Gennaio 2024
Data fine	Esperienza attuale
Luogo	Napoli, Via Argine
Sito web	www.hitachirail.com
Ruolo	Consulente tramite Teoresi S.p.A.

Ho lavorato presso Hitachi come consulente Teoresi, in svariati progetti.

In particolare, ho lavorato al progetto 360Flow, lato veicolo, per la manutenzione dei moduli già implementati oltre allo sviluppo di nuove librerie e moduli software.

In Giugno 2024, sono stato invece assegnato al progetto 360Motion, terminato invece verso Ottobre 2025.

Per un approfondimento dei progetti e delle tecnologie coinvolte, si può far riferimento alle apposite sezioni.

2 Tecnologie e strumenti

Di seguito, l'elenco delle varie tecnologie e strumenti utilizzati nella mia esperienza.

2.1. Linguaggi di programmazione

2.1.1. Bean Shell

Progetti (1): 360Motion

Ho utilizzato il linguaggio BeanShell per le attività di test automation del progetto *360Motion*, basato sull'uso del software Apache JMeter.

Quindi linguaggio utilizzato solamente per scripting, e non a livello di progetto.

2.1.2. Java

Progetti (4): NegozioPerfetto, MagazzinoPerfetto, MBE LAB, 360Motion

Nella mia esperienza ho utilizzato Java in più di un progetto, ma principalmente in due modi:

1. Sviluppo di applicazioni mobile in ambito logistico, quindi sfruttando il framework previsto dallo sviluppo **Android**. Per ulteriori dettagli relativi alle mie esperienze Android, si può far riferimento alla apposita sezione.
2. Sviluppo di micro servizi in Java tramite framework Spring Boot.

Lato progetto, ho utilizzato sia Maven che Gradle come strumento di gestione delle dipendenze, del versionamento, ecc.

2.1.3. JavaScript

Progetti (5): Movitel, CRM-O, ApriUnTicket, FantaRegolamento, MyPortFolio

Ho utilizzato **JavaScript** in diversi progetti lato *front-end*, in particolare nelle mie prime esperienze lavorative, dove ero responsabile dello sviluppo e della manutenzione di alcuni applicativi web.
In questa fase, l'utilizzo di JavaScript è stato di tipo tradizionale, in combinazione con **HTML** e **CSS**, senza l'impiego di framework particolari, per realizzare le funzionalità e l'interfaccia desiderate.
Successivamente ho utilizzato anche **jQuery**, per semplificare l'accesso e la manipolazione degli elementi del DOM.
Infine, in alcuni progetti personali, ho avuto modo di approfondire l'uso di **React**,

implementando un'applicazione basata su **JSX (JavaScript XML)** per raggiungere gli obiettivi dell'applicazione.

Ho anche avuto modo di esplorare lo Unit Testing in Javascript, sfruttando il framework **Jasmine**.

2.1.4. Node.js

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Ho utilizzato **Node.js** principalmente per la scrittura di script e funzioni serverless in **AWS Lambda**, sfruttando la sua capacità di gestire in modo efficiente **eventi e richieste asincrone**.
La mia esperienza con questo linguaggio è quindi più orientata allo **scripting operativo** che allo sviluppo di applicazioni complete: ho scritto codice per connettermi a database, elaborare dati e integrare servizi cloud, ma non mi sono occupato di aspetti come build, dipendenze o configurazioni complesse di progetto.

Le Lambda potevano essere scritte anche in **Python**, ma in alcuni casi si preferiva **Node.js** per la sua rapidità nel gestire operazioni I/O asincrone.
Ad esempio, ho realizzato una funzione che riceveva richieste API, interrogava un database **DynamoDB**, e restituiva i risultati in formato JSON in modo non bloccante, sfruttando le **Promise** e le **funzioni async/await** di Node.js.

2.1.5. Python

Progetti (5): NegozioPerfetto, MagazzinoPerfetto, MBE LAB, TCR, 360FlowVehicle

Nella mia esperienza ho utilizzato Python in più di un progetto, con le seguenti modalità:

1. In ambito AWS, ho utilizzato Python per l'implementazione delle funzioni Lambda, che venivano poi esposte tramite le apposite API Gateway previste sempre nell'ambito AWS. In particolare, queste funzioni Lambda sono state sviluppate con l'utilizzo della libreria **boto3**, utile per l'interazione con gli altri servizi AWS (esempio database, storage, ecc.).

2. Nel progetto 360FlowVehicle, Python era il linguaggio di programmazione con cui l'intero progetto è stato sviluppato. I moduli sviluppati venivano poi containerizzati in ambito Docker. In questo contesto è stato utilizzato anche un server **Flask** per simulare richieste HTTP.

3. Nel progetto TCR, mi sono occupato della migrazione del progetto da Python 2.7 a Python 3.9, andando a gestire anche casistiche in cui alcune librerie utilizzate dalla 2.7 non erano più supportate dalla versione 3.9.

2.1.6. Swift

Progetti (1): QL-Mobile

Ho utilizzato il linguaggio **Swift** per lo sviluppo dell'app mobile *QL-Mobile*.

Si tratta dell'unico progetto in cui ho impiegato questo linguaggio, ma l'esperienza mi ha permesso di acquisire familiarità sia con **Swift** sia con l'**IDE Xcode** di Apple.

Ho approfondito le **best practice di sviluppo iOS**, realizzando un'interfaccia fluida e funzionale, integrando **API esterne** e gestendo un **database locale** per la memorizzazione di dati persistenti, come le impostazioni utente.

2.1.7. VB.NET

Progetti (3): Movitel, CRM-O, ApriUnTicket

Ho avuto modo di utilizzare il linguaggio **VB.NET** nei primi anni della mia esperienza lavorativa, occupandomi inizialmente della manutenzione di applicativi già in produzione al momento della mia assunzione.
Con il tempo, ho esteso le mie competenze sviluppando **nuove funzionalità** e partecipando all'**evoluzione dei moduli esistenti**, migliorandone prestazioni e stabilità.
Questa esperienza mi ha permesso di approfondire la **programmazione ad oggetti in ambiente .NET**, l'interazione con **database SQL Server** e la gestione del **ciclo di vita del software** in contesti aziendali strutturati.

2.2. Framework & Tooling

Di seguito, l'elenco dei vari framework e tooling che ho utilizzato nelle mie esperienze.

2.2.1. Android – sviluppo mobile

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Il framework Android è stata la tecnologia che ho usato per circa 7 anni, dal 2016 al 2023.

Come IDE di sviluppo è stato usato Android Studio, mentre come linguaggio ho usato per lo più Java, ma ho avuto modo di usare anche Kotlin in altri progetti minori e per funzioni specifiche.

Ho avuto modo di adottare architetture di tipo **MVC** (Model-View-Controller) o **MVP** (Model-View-Presenter) per garantire una separazione chiara delle responsabilità: le activity fungevano da model o presenter, che invocano fragment che fungono da view, per mostrare l'interfaccia utente e reagire agli input dell'utente.

Ho organizzato i file XML per definire il layout dell'interfaccia utente e ho utilizzato i file di risorse per la gestione delle stringhe, delle dimensioni e degli stili.

L'associazione degli elementi della view ed il relativo controllore veniva eseguita tramite view binding.

Inoltre, ho utilizzato Gradle come strumento di build per gestire le dipendenze esterne e assicurare una corretta configurazione del progetto.

Riguardo i database, ho usato sia database locali SQLite sia progetti in cui i dati venivano rilevati tramite API.

Nei progetti NegozioPerfetto, MBE LAB e MagazzinoPerfetto, questo database locale veniva poi sincronizzato con un database in cloud tramite un processo scritto ad-hoc.

Altri servizi usati in questo contesto sono Firebase, usato per ricevere notifiche di crash o analisi e per inviare notifiche push, e Git, usato per il versioning.

Per quanto riguarda invece l'invocazione di API, si enuncia anche l'utilizzo di Retrofit.

2.2.2. Git

Ho utilizzato **GIT** come strumento di versioning in tutti i progetti in cui ho lavorato.

La mia esperienza mi ha portato a utilizzare GIT in quasi tutte le sue funzionalità, lavorando su branch separati che venivano poi pushati sia tramite pull-request sia tramite commit diretti sul branch **master**.

Ho anche utilizzato la variante **Git Flow**, che permette di seguire un modello di branching strutturato e standardizzato, gestendo in modo chiaro e sicuro le fasi di sviluppo, release e hotfix.

Questo approccio facilita il lavoro in team, riduce il rischio di conflitti e permette di mantenere sempre stabile il branch principale.

2.2.3. Gradle

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Ho utilizzato Gradle in vari progetti, in particolare nei progetti implementati tramite framework Android. La tecnologia veniva utilizzata ovviamente per la gestione delle dipendenze, il versioning dell'applicativo e la configurazione dei task di build, semplificando così il processo di compilazione e distribuzione dell'app.

2.2.4. Maven

Progetti (1): 360Motion

Per il progetto 360Motion ho utilizzato Maven per la gestione centralizzata delle dipendenze e il versionamento dei vari servizi. Il sistema di build supportava, per alcuni servizi, output multipli, includendo sia un applicativo eseguibile sia una libreria, ottimizzando compilazione e distribuzione.

2.2.5. Spring Boot

Progetti (1): 360Motion

Durante lo sviluppo del progetto *360Motion*, ho contribuito alla realizzazione di un sistema basato su **Spring Boot** e **architettura a microservizi**, dove ciascun servizio era indipendente e comunicava con gli altri tramite **Kafka**.

Ho curato la definizione delle **API REST**, la gestione del **data layer** tramite **Spring Data JPA** e **Hibernate**, e la configurazione dei database **PostgreSQL** e **Redis** per la memorizzazione dei dati.

L’intero progetto è stato gestito e costruito con **Maven**, assicurando una solida integrazione tra i vari componenti.

2.3. Cloud Services

2.3.1. Amazon Web Services - AWS Amazon Cloud

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Ho utilizzato i servizi AWS durante le mie esperienze di sviluppo mobile Android.

Di seguito i servizi usati che la piattaforma mette a disposizione:

2.3.1.1. Aurora DB

Gestione di un database relazionale basato su **MySQL**.

In un contesto **AWS**, i dati venivano interrogati esclusivamente tramite **Lambda**, che eseguivano le query, trasformavano i dati e li restituivano al chiamante.

Questo approccio ha permesso di mantenere sicuro l'accesso al database e di garantire una gestione scalabile e serverless delle operazioni di lettura e scrittura.

2.3.1.2. Dynamo DB

Gestione di un **database non relazionale serverless** basato su struttura **key-value**.

Questa soluzione è stata utilizzata per la storicizzazione di dati che non era possibile gestire in un database relazionale, come ad esempio documenti **JSON**.

Per alcune entità, infatti, uno degli attributi conteneva un JSON; per queste entità è stata quindi scelta la storicizzazione su **DynamoDB** anziché su Aurora, garantendo maggiore flessibilità e scalabilità nella gestione dei dati.

2.3.1.3. Lambda

Servizio di elaborazione che consente di eseguire del codice senza la necessità di effettuare il provisioning o la gestione dei server.
Questo codice storicizzato è poi invocabile tramite API o tramite SDK Amazon.
Il codice per le funzioni di lambda può essere scritto con una lista di linguaggio che la piattaforma mette a disposizione, e nello specifico ho usato: Python e NodeJS.

Le lambda sviluppate, in genere, avevano la responsabilità di accedere ai database previsti, estrarre dati, trasformarli e restituirli al chiamante tramite API.

2.3.1.4. API Gateway

Servizio AWS per la creazione, la pubblicazione, la gestione, il monitoraggio e la protezione di API REST, HTTP e WebSocket a qualsiasi livello.
Gli sviluppatori di API possono creare API in grado di accedere ad AWS o ad altri servizi Web, nonché ai dati archiviati nel cloud AWS.
In particolare, nella mia esperienza, ho sempre provveduto a collegare una API ad una funzione Lambda specifica.

2.3.1.5. Cognito

Piattaforma di identità per app Web e per dispositivi mobili, utile a gestire l'accesso al sistema tramite credenziali.
È una directory utente, un server di autenticazione e un servizio di autorizzazione per i token di accesso OAuth 2.0 e le credenziali AWS.

2.3.1.6. S3 (Amazon Simple Storage Service)

Servizio Amazon dedicato alla archiviazione e gestione di una serie di documenti e file generici previsti dal sistema in implementazione.
Ho gestito nel tempo sia bucket privati che pubblici, in base alla richiesta utente.

2.3.1.7. Cloud Watch

Servizio di monitoraggio di AWS per le risorse cloud e le applicazioni in esecuzione su AWS.
Ho utilizzato Amazon CloudWatch per raccogliere e tenere traccia dei parametri, raccogliere e monitorare file di log prodotti dai vari servizi gestiti nel tempo.
Esempio: visualizzare i log prodotti da una specifica funzione lambda.

2.3.1.8. Batch

Servizio AWS utilizzato per eseguire processi di lunga durata.
Ho utilizzato batch per eseguire, ad esempio, calcoli notturni statistici che in altre condizioni avrebbero impiegato troppo tempo e risorse.

Questi job potevano essere schedulati (esempio ogni notte) o eseguiti su richiesta (esempio l’esportazione in CSV dei dati di magazzino).

2.3.1.9. ECR (Amazon Elastic Container Registry)

Servizio di registro delle immagini container.
Le immagini che ho caricato in ECR erano sviluppate tramite Docker e poi caricate in ECR come repository private.
Tale servizio è stato usato per poter avviare le suddette immagini Docker tramite AWS Batch.

2.4. Container & Orchestration

2.4.1. Docker

Progetti (4): MagazzinoPerfetto, MBE LAB, 360Flow, 360Motion

Nella mia esperienza ho lavorato in maniera massiccia con **Docker**, utilizzando questa tecnologia in due modalità principali:

1. **Sviluppo di app mobile con integrazione AWS**: ho utilizzato Docker per creare container a partire da script sviluppati in **PHP**. Le immagini generate venivano poi caricate su **AWS** tramite il servizio **AWS Batch**, utilizzato per eseguire job schedulati e non, consentendo di processare operazioni anche di lunga durata, che potevano richiedere diverse ore.

2. **Progetti Hitachi 360Flow e 360Motion**: ho impiegato Docker nella sua funzione principale, ossia la containerizzazione di servizi Java, permettendone l'esecuzione sia in locale sia in ambiente di produzione tramite **Kubernetes**. In questo contesto ho utilizzato **Docker Compose** per buildare e orchestrare i vari container in maniera efficiente.

2.5. Database

Nella mia esperienza professionale, ho lavorato con diversi tipi di database, sia relazioni che non relazionali.

2.5.1. Database Relazionali

2.5.1.1. MySQL

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Database usato nei progetti NegozioPerfetto, MagazzinoPerfetto e MBE LAB come database in cloud per la storicizzazione di dati provenienti dalle APP Android.

Gli strumenti utilizzati sono stati:

- Trigger
- Stored function
- Stored procedure
- Indici

Nel progetto MagazzinoPerfetto, sono stati implementati trigger che invocavano chiamate API che a loro volta invocavano procedure lambda in AWS, per rendere disponibili web hook utili all'utente finale.

Questa soluzione ha consentito l'integrazione fluida con altre applicazioni e sistemi esterni, garantendo una gestione efficiente delle notifiche e degli aggiornamenti in tempo reale all'interno del magazzino.

2.5.1.2. SQL Server

Progetti (2): ApriUnTicket, CRM-O

Database usato nei progetti CRM-O ed ApriUnTicket, come soluzione usata negli applicativi web menzionati.

2.5.1.3. PostgreSQL

Progetti (2): 360Flow, 360Motion

Nell'ambito dei progetti Hitachi, ho utilizzato PostgreSQL come database per la storicizzazione dei dati elaborati dal sistema. Anche per questo database sono state

utilizzati trigger e procedure. Inoltre, per alcune entità, sono stati anche gestiti JSON come colonne di un record.

2.5.2. Database No SQL

2.5.2.1. Dynamo DB

Progetti (1): NegozioPerfetto

Amazon DynamoDB è un servizio di database NoSQL gestito, offerto da AWS, progettato per garantire prestazioni veloci e scalabili. È altamente disponibile, supporta dati strutturati in tabelle, ed è ottimizzato per applicazioni che richiedono basse latenze e alta velocità nelle operazioni di lettura e scrittura.

Nel progetto NegozioPerfetto, è stato usato DynamoDB per la storicizzazione di alcuni tipi di entità, come ad esempio documenti JSON da associare a specifiche transazioni di magazzino storicizzate, invece, in MySQL.

2.5.2.2. Redis

Progetti (1): 360Motion

Redis è un database in-memory open-source che funziona come un data-store NoSQL. È noto per la sua velocità estremamente elevata, poiché i dati vengono memorizzati principalmente in memoria (RAM) anziché su disco. Redis supporta strutture dati come stringhe, liste, set, hash e bitmaps, ed è spesso utilizzato per caching, session management, real-time analytics e messaging.

Nel progetto 360Motion, ho usato Redis per la storicizzazione e lettura di JSON che dovevano essere letti e modificati di continuo in runtime.

Usato, inoltre, **RedisCommander** come strumento per l'accesso a tale database.

2.6. Messages Brokers

2.6.1. MQTT

Progetti (2): 360Flow, 360Motion

MQTT è un protocollo di messaggistica leggero basato su **publish/subscribe**, progettato per reti con larghezza di banda limitata e connessioni intermittenti.

Nei progetti **360Flow Veicolo** e **360Motion**, ho utilizzato MQTT per consentire ai veicoli in movimento di inviare ai componenti centralizzati dati come **vehicle-congestion**, **vehicle-position** e simili.

Questo protocollo è stato scelto perché, ad alte velocità, la connettività potrebbe non essere garantita; MQTT è progettato proprio per garantire la comunicazione anche in presenza di connessioni instabili o di scarsa qualità.

2.6.2. Apache Kafka

Progetti (1): 360Motion

Ho utilizzato **Apache Kafka** nel progetto **360Motion** per distribuire messaggi tra i diversi **consumer** dello streaming interno.

Per la gestione e il monitoraggio dei topic e dei messaggi in circolazione, ho utilizzato **AKHQ**, che ha permesso di leggere e analizzare facilmente i flussi di dati all'interno della piattaforma Kafka.

2.7. Pattern

Qui di seguito elencherà i vari pattern usati e padroneggiati nel tempo.

2.7.1. Singleton pattern

Pattern creazione usato per gestire oggetti che dovrebbero essere istanziati una sola volta nel sistema.

Usato in quasi tutti i progetti in cui ho lavorato.

2.7.2. Factory pattern

Il Factory pattern è un pattern creazionale che consente di creare oggetti senza specificare esplicitamente la classe concreta di ogni oggetto.

Questo è possibile attraverso l'utilizzo di un'interfaccia o di una classe astratta che definisce un metodo per la creazione di oggetti.

Le classi concrete che implementano questa interfaccia decidono quale tipo di oggetto istanziare, in base alle esigenze dell'applicazione.

Questo pattern è ampiamente utilizzato nei progetti 360Flow e 360Motion per gestire la creazione dinamica di vari tipi di oggetti in modo flessibile e scalabile, permettendo una migliore separazione delle responsabilità e facilitando l'estensione del sistema con nuovi tipi di oggetti.

2.7.3. Observer pattern

Il pattern Observer è un pattern comportamentale che permette a un oggetto, chiamato "soggetto", di notificare automaticamente gli "osservatori" registrati quando avviene un cambiamento nel suo stato. Gli osservatori sono quindi aggiornati e possono reagire di conseguenza.

Questo pattern è stato usato nel progetto 360Flow in modo da mettere in ascolto alcuni componenti dei vari moduli sui subscriber MQTT previsti, in modo da avere un solo componente in ascolto sul broker MQTT e tante classi che sono in ascolto sull’aggiornamento dei valori osservati.

2.7.4. Mediator pattern

Il pattern Mediator è un pattern comportamentale che favorisce la comunicazione indiretta tra gli oggetti, promuovendo la debole dipendenza tra di essi.

Un oggetto "Mediatore" facilita la comunicazione tra gli oggetti "Colleghi" attraverso una logica centralizzata, riducendo così il numero di connessioni dirette tra gli stessi.

Questo pattern è stato utilizzato nel progetto 360Flow per facilitare la comunicazione tra componenti diversi del sistema.

2.8. Documentazione & PM

Di seguito vari strumenti usati per documentare i software a cui ho lavorato.

2.8.1. Doors

Progetti (1): 360Motion

IBM Engineering Requirements Management DOORS è uno strumento di gestione dei requisiti ampiamente utilizzato nel settore dell'ingegneria e dello sviluppo di software.

Ho utilizzato questo strumento per gestire i requisiti del progetto 360 Motion.

2.8.2. Gestione task & Attività

Gli strumenti utilizzati per la gestione di attività interne sono stati:

- **Planner**
  Microsoft Planner è uno strumento di gestione dei progetti e delle attività integrato in Microsoft 365.

- **Project Place**
  Project Place è una piattaforma di gestione dei progetti e delle collaborazioni online, sviluppata da Planview.

2.9. Hardware & IoT Integration

2.9.1. ZPL

Progetti (3): NegozioPerfetto, MagazzinoPerfetto, MBE LAB

Ho utilizzato lo ZPL per creare etichette stampabili tramite stampanti Zebra, consentendomi di gestire le dimensioni delle etichette e altri aspetti legati alla funzionalità. Questo linguaggio si è dimostrato particolarmente utile per le stampe via Bluetooth, riducendo il carico di dati da inviare dal dispositivo alla stampante, ottimizzando così il processo di stampa.

2.9.2. RFID

Progetti (1): MagazzinoPerfetto

Nel contesto del progetto MagazzinoPerfetto, ho applicato la tecnologia RFID per facilitare il tracciamento dell'entrata e dell'uscita della merce, nonché per semplificare le attività di inventario e il controllo dello stock in magazzino.

In particolare, abbiamo utilizzato l'area di memoria EPC per memorizzare i dati univoci degli articoli, mentre il TID è stato sfruttato per monitorare il numero di rilevazioni dell'EPC specifico durante la lettura RFID.

L’integrazione RFID con l’APP è stata eseguita tramite SDK di Zebra.

3 Progetti

3.1. 360 Motion

Azienda	Hitachi Rail - Teoresi S.p.A.
Sito web	https://www.youtube.com/watch?v=3Oe-XVIZd-I

360Motion è una piattaforma pionieristica che sta trasformando la mobilità urbana consolidando i dati provenienti da diversi servizi di trasporto in una soluzione unificata. È alimentato da algoritmi avanzati di intelligenza artificiale e ML, che gli consentono di ottimizzare dinamicamente le reti di trasporto pubblico, facilitare la pianificazione degli scenari e anticipare e mitigare le interruzioni.

In pratica, 360Motion riunisce i dati di più servizi di trasporto (bus, metro e veicoli vari) in un'unica piattaforma unificata.

Questi dati vengono ricevuti ed analizzati, con lo scopo di fornire una piattaforma web che possa fornire all'utente quante più informazioni e operatività possibili.

360Motion è una piattaforma indipendente dall'hardware, in grado di scalare le soluzioni di mobilità esistenti raccogliendo e acquisendo dati da qualsiasi tipo di fonte e consentendo a operatori e passeggeri di comprendere il modo ottimale di spostarsi attraverso i paesaggi urbani.

Tecnicamente, invece, 360Motion è un progetto a micro-servizi, scritti in Java (SpringBoot), Python, ed una parte front-end scritta in Angular, che viene distribuito tramite Kubernetes.

In questa mia esperienza come consulente, le tecnologie utilizzate sono state:

- Java con SpringBoot, IntelliJ Idea come IDE
- Maven come strumento di gestione
- PostGress, con PgAdmin come strumento di accesso e gestione.
- Docker
- Swagger
- Git, come strumento di versioning
- Kafka, come distributore interno di messaggi, ed AKHQ come strumento di lettura dei vari topic.
- Redis, ossia un database NoSQL in cache per ottimizzarne le letture. Usato RedisCommander come strumento per accedervi.
- Azure DevOps, come strumento di accesso ai repository e esecutore di pull-request su main, pipeline per compilazione e rilascio automatico.

- SonarQube, strumento per eseguire analisi statica del codice
- JMeter per l'attività di verifica e fix dei work-item rilevati, con linguaggio BeanShell
- Doors, strumento usato come repository dei requisiti di sistema, ove eventualmente applicare modifiche ad essi.
- Project Place, strumento per la gestione dei task interni.
- Cisco Any Connect, usato per accesso a VPN interne.

Invece, le attività di cui mi sono occupato sono:

- Implementazione di Integration-test per controller Spring che ne erano sprovvisti, con JUnit.
- Incremento della code coverage del codice.
- Integrazione di Swagger nei controller previsti.
- Verifica ed eventuale fix per work-item rilevati dal team di V&V (Verification & Validation)
- Refactoring di un SDK interno utile alla gestione delle proprietà che i singoli componenti potevano richiedere al database
- Implementazione della libreria **Test Core**, utilizzata come framework per la scrittura di test di integrazione.
  Questa libreria mette a disposizione vari connettori (database, Redis, Kafka, ecc.) e gestisce, tramite il file **application. properties**, l'accesso a tali connettori tramite test container o tramite l'ambiente reale locale.
  I connettori vengono implementati utilizzando il **Factory pattern**, garantendo così una facile aggiunta di nuovi elementi.
- Implementazione di un nuovo servizio Java, **platform-integration-test**, che, tramite la libreria Test Core, contiene una serie di test di integrazione per verificare il comportamento dei vari microservizi rilasciati. Questo strumento è stato impostato in modo da eseguire, ogni notte, una suite di test di integrazione definiti tramite JSON, in modo da verificare, ogni giorno, l'impatto di nuovi sviluppi o modifiche all'ambiente sulla stabilità del software.

Nel contesto di IntelliJ, sono stati usati i seguenti plugins:

- Lombok, usato per importare annotazioni e ridurre il codice scritto.
- SonarLint, utile a rilevare eventuali problemi del codice.

3.2. 360 Flow – Veicolo

Azienda	Hitachi Rail - Teoresi S.p.A.
Sito web	www.hitachirail.com

360Flow è un sistema a micro servizi per il conteggio delle persone a bordo veicolo.

Il sistema è stato sviluppato in Python su più moduli che comunicano tra di loro tramite algoritmo MQTT e tramite l'esterno tramite HTTP e socket.

Il sistema si interfaccia con la sensoristica per la manipolazione dei dati di congestionamento a bordo treno, dei dati di posizionamento e dei dati di diagnostica.

I moduli sono sviluppati per container Docker su sistema operativo Windows. Il sistema utilizza dei sistemi per difendersi da attacchi informatici. Inoltre è presente la possibilità di interfacciamento diretto con il TCMS a bordo treno tramite un modulo sviluppato in C con protocollo TRDP.

Inoltre è presente un modulo di AI per la rilevazione delle porte aperte o chiuse in caso di impossibilità di connessione con il TCMS. È stato sviluppato un proxy per consentire la connessione da parte di app terze, come ad esempio HMI per di manutenzione che permette l'invio di comandi al sistema o la lettura di dati da esso. È presente anche un database di tipo relazionale PostgreSQL il quale viene utilizzato per la configurazione dei dispositivi e per la storicizzazione dei dati. Si è dovuta gestire tutta la parte relativa alla sicurezza informatica per le comunicazioni in ingresso ed in uscita dal sistema, in particolare utilizzando certificazioni sulle comunicazioni, crittazione dei dati con chiavi simmetriche e asimmetriche e combinazioni utente password per l'autenticazione.

Tra i dati che la 360Flow raccoglie possiamo nominare:

- Numero di persone entrate ed uscite ad una certa stazione per una certa porta del treno
- Numero di persone che, durante il viaggio, ha cambiato la propria carrozza di viaggio.
- Tipo di oggetti che occupano il treno, esempio biciclette, carrozzine, ecc.

360Flow prevede software sia per la parte veicolo (treno) che per la parte di stazione.

In particolare, ho lavorato alla parte di veicolo, eseguendo i seguenti sviluppi:

- Implementazione delle operazioni CRUD sul database PostgreSQL.
- Gestione del formato JSON dei messaggi che il modulo ActionCore (proxy) doveva prevedere.

- Gestione dell'algoritmo di crittazione che doveva essere applicato ai messaggi da pubblicare tramite MQTT.
- Sviluppo della libreria centralizzata per il logging del sistema.
- Sviluppo di un componente del modulo di diagnostica, il FileArchiver, utile a gestire i log creati nel punto precedente. Di fatti, il FileArchiver verifica se i log abbiano superato una certa dimensione oppure una certa data di creazione, ed in tal caso vengono archiviati ed inviati al sistema esterno centrale.
- Sviluppo del modulo DiagnosticReporter, modulo dedicato al monitoraggio di alcuni valori del sistema centrale.
  Esempio, se la RAM della macchina a bordo treno supera alcuni limiti previsti, il sistema di controllo centrale deve essere allertato tramite messaggi HTTP e/o MQTT.

Le **tecnologie** e strumenti utilizzati sono stati:

- **Python** come linguaggio di programmazione dei moduli, e **PyCharm** come IDE di sviluppo.
- **Docker** per la creazione di immagini per ogni modulo previsto, così da rendere il funzionamento di un container (modulo) indipendente dal funzionamento degli altri.
- **CodeCommit + GIT** per il versioning del progetto.
  Il team di sviluppo era composto da 7 persone lato veicolo, che potevano interagire anche sugli stessi file.
  In particolare, è stato usato **Git Flow** per assolvere alle funzionalità di versioning.
- **MQTT** per lo scambio di messaggi tra i moduli.
  Essendo questo un software da installare su macchina a bordo treno, la connessione poteva essere instabile, ecco perché si è scelto di usare questo protocollo anziché l'HTTP per lo scambio di messaggi.
- **PostgreSQL** come database per i dati raccolti. È stato selezionato PostgreSQL come database a causa della sua capacità di gestire un alto volume di scritture simultanee, necessario per il sistema.

3.3. TCR - Test Case Runner

Azienda	Progetto interno presso Teoresi S.p.A.

Il Test Case Runner è una piattaforma generica per implementare l'automazione dei test, basata su un'applicazione web **Python Django**.

Le funzionalità principali consentono di:

- caricare una suite di test basata su un formato **XML** validato
- caricare plugins su richiesta
- memorizzare su un database **SQLite** le informazioni del test (ovvero test, descrizione, requisiti, report di esecuzione)
- eseguire le fasi di prova
- esportare documenti personalizzati in base alle informazioni archiviate nel database.

L'applicazione può essere personalizzata per testare un progetto specifico.

Il core carica automaticamente i plugins presenti in una cartella specifica.

I plugin possono implementare le seguenti interfacce:

- pre-esecuzione: una serie di passaggi eseguiti prima dell'esecuzione dei passaggi di test (ovvero preconfigurazione del test)
- eseguire: una serie di passaggi di test
- post-esecuzione: una serie di passaggi eseguiti dopo l'esecuzione dei passaggi di test (ovvero smontaggio)

L'applicazione web Python Django consente di:

- caricare una suite di test
- creare una nuova suite di test
- modificare una suite di test (ad esempio aggiornando titolo o descrizione, aggiungendo test case, modificando passaggi di test, …)
- eseguire una suite di test
- mostra la cronologia della suite di test
- Esportare una suite di test.

In questo progetto mi sono occupato di:

- Migrazione del progetto da **Python 2.7** a **Python 3.9**, in cui sono stato coinvolto nel refactoring di alcune funzionalità, dato che alcune librerie usate in Python 2.7 non erano presenti per Python 3.9 oppure producevano risultati diversi. Nomino ad esempio la libreria py_spy, sostituita con py_shark per lo sniffing di pacchetti di rete UDP.
- Integrazione del protocollo **TRDP**, oltre a quello già presente **IPTCOM**.
  Il TRDP (Train Real-time Data Protocol) è un protocollo di comunicazione progettato specificamente per il settore ferroviario, che permette la trasmissione di dati in tempo reale tra i vari sistemi di un treno. È stato sviluppato per soddisfare le esigenze di comunicazione affidabile e deterministica in ambienti ferroviari, dove è fondamentale avere un basso ritardo e alta disponibilità dei dati.
- Definizione di una **GUI** utile alla creazione dei test XML in maniera rapida ed intuitiva.

Le **tecnologie** e strumenti utilizzati sono stati:

- Git (con **Bitbucket**) come sistema di versioning.
- **JIRA** per la gestione del progetto con Agile Scrum.
- **Confluence** per la gestione della documentazione.
- **Python Django** for the Web application.

3.4. Negozio Perfetto

Azienda	Advise SRL
Sito web	http://www.negozioperfetto.it/
Google Play Store	NegozioPerfetto APP

NegozioPerfetto è un progetto nato dall'idea di aiutare i piccoli negozianti a contrastare le grandi multinazionali che oggi dominano il mercato.
NegozioPerfetto è quindi un gestionale light, sviluppato per essere quanto più intuitivo e semplice possibile.
Oltre alle classiche funzioni di gestione della movimentazione di magazzino (quindi carico, scarico, trasferimenti, ecc.), l'obiettivo del progetto era quello di implementare un sito e-commerce completamente ed automaticamente integrato al gestionale, cosi da mostrare online sempre anagrafiche e stock aggiornati in tempo reale.
Altra questione affrontata nel progetto era l'utilizzo della APP in offline: dato che, in un negozio, la connessione potrebbe essere scarsa o comunque non perfetta, il negoziante avrebbe potuto tranquillamente vendere.
Tramite poi un processo di sincronizzazione, i dati venivano sincronizzati sia in entrata che in uscita con il cloud, permettendo quindi sia un backup dei dati sia la possibilità di lavorare offline con una lista indefinita di dispositivi.

In questo progetto sono stato coinvolto nelle seguenti fasi del software:
- Analisi
- Design
- Implementazione
- Manutenzione.

Di seguito, invece, le tecnologie usate:

Amazon AWS Cloud

Come detto nella descrizione del progetto, la APP prevede di lavorare in offline per poi sincronizzare le modifiche (sia in importazione che esportazione) su un server in cloud.
È stato scelto AWS come piattaforma cloud, di cui sono stati utilizzati svariati servizi:

- **EC2**: servizio per avere in esecuzione un server Linux.
- **Aurora DB**: database relazione MySQL.
- **DynamoDB**: database non relazione usato per contenere alcune informazioni particolari.
- **Lambda**: servizio con cui creare delle procedure online raggiungibili tramite API. Per la scrittura di queste lambda, sono stati usati principalmente due linguaggi: **Python** e **NodeJS**.
- **API Gateway** : servizio con cui creare delle API associate poi a delle funzioni lambda.
- **Cognito** : servizio con cui implementare velocemente le procedure di login, recupero password e registrazione al sistema.
- **S3** : spazio di storage in cui poter salvare file.
- **Iot Core** : servizio di IOT usati per richieste verticali, esempio implementazione di un Dash button con cui ordinare uno specifico articolo solamente cliccando il tasto previsto.

Java (Android)

La APP è stata scritta utilizzando il linguaggio Android nativo, mentre come IDE è stato usato Android Studio.
È stato implementato un database in locale SQLite che poi, tramite API, veniva sincronizzato con il database in cloud in MySQL.
Altri strumenti usati durante il progetto sono stati Fabric per l'Analytics e Crashlitycs e Firebase per l'invio di notifiche push al sistema.

Python, NodeJS

Linguaggi utilizzati per scrivere le funzioni lambda necessarie al sistema.

Git Hub

È stato usato GitHub come piattaforma di versioning.

WooCommerce

Il sito integrato al progetto è stato creato tramite piattaforma WooCommerce, ed

ovviamente sono state implementate tutte le API utili al progetto per implementare l'integrazione con il sito discussa nella descrizione del progetto.

MySQL

Come database in cloud è stato scelto una soluzione basata sul MySQL.
Negli anni sono stati approfonditi varie tematiche relativamente al database, ossia sono state implementati trigger, stored functions, stored procedures ed indicizzazione delle tabelle più sollecitate dalle operazioni previste.

ZPL

Linguaggio utilizzato per creare etichette stampabili tramite stampanti Zebra, in modo da gestire dimensione etichette ed altri aspetti connessi alla funzionalità. Particolarmente utile per stampe Bluetooth, in modo da alleggerire il carico da inviare da dispositivo a stampante.

3.5. MBE LAB

Azienda	Advise SRL
Sito web	www.mbe.it
Google Play Store	MBE LAB APP

MBE LAB è un gestionale per logistiche nato dall'interesse di Mail Boxes Etc. verso NegozioPerfetto.
Essendo però NegozioPerfetto pensato per negozianti e più concentrato sulla vendita al banco, ecco che viene avviato questo nuovo progetto di logistica.
Se da un lato l'infrastruttura non ha subito grosse modifiche rispetto a NegozioPerfetto (incremento del numero delle tabelle, campi diversi), la APP è stata completamente rivisitata, dato che i requisiti tra la vendita al banco e la logistica sono parecchio differenti.
Anche sotto l'aspetto delle tecnologie il sistema non ha subito grosse modifiche.

Dunque, le differenze tra i due progetti sono per lo più funzionali che infrastrutturali.
In questo progetto ci siamo scontrati con l'enorme mole di dati che logisticamente venivano prodotto, e ciò ha comportato alcune migliorie al database ed alla APP stessa in termini di performance.
Basti pensare che, in alcuni casi particolare, la APP ha dovuto gestire, sincronizzando automaticamente più di 50 terminali, le emissioni di circa 10.000 documenti di spedizione al giorno.
Le funzionalità si sono poi concentrato sull'integrazione tra i sistemi da noi gestiti e quelli di MBE, implementando una lista corposa di API (lettera vettura, spedizioni, utenza) ed un sistema di SSO (single sign-on).

MBE LAB è un progetto utilizzato in vari paesi del mondo, quali Italia, Germania, Inghilterra, Spagna, Polonia, Kazakistan, Panama e Francia, e la lista con il tempo potrà solamente aumentare.
In questo progetto sono stato coinvolto nelle seguenti fasi del software: Analisi, Design, Implementazione e manutenzione.

3.6. Magazzino Perfetto

Azienda	Advise SRL
Sito web	www.magazzinoperfetto.it
Google Play Store	MagazzinoPerfetto APP

Dopo l'esperienza fatta con MBE LAB, dunque, si arriva al lancio di un nuovo progetto: MagazzinoPerfetto.
MagazzinoPerfetto nasce anch'esso come gestionale, ma questa volta non di negozi, non di logistiche, ma magazzini nel loro completo ciclo di vita.
Quindi, anche in questo caso, le modifiche di infrastruttura sono ben poche (incremento del numero delle tabelle, campi diversi), ma sono maggiormente modifiche funzionali ed integrazioni verticali verso altri sistemi.
Infatti, avendo dovuto implementare un WMS completo, ci siamo ritrovati a doverci interfacciare con molteplici sistemi di terzi parti, in base alle esigenze dei clienti.

Tra le principali integrazioni eseguite:

- Integrazione con un sistema di blockchain, con cui memorizzare le transazioni della merce.
- Implementazione della tecnologia RFID.
- Integrazione con sistemi di contabilità.
- Integrazione con la dogana francese.
- Integrazione totale con Zebra, azienda di hardware di magazzino (terminali, RFID, visori).
- Integrazione con WooCommerce per siti customizzati.

In questo progetto sono stato coinvolto nelle seguenti fasi del software: Analisi, Design, Implementazione e manutenzione.

3.8. Apri Un Ticket

Azienda	Advise SRL

ApriUnTicket è un CRM (Customer Relationship Management) sviluppato dapprima per uso interno aziendale, e poi venduto a clienti terzi per alcune funzionalità utili individuate in esso.
Questo CRM, a dispetto degli altri, è un CRM sviluppato con l'obiettivo di facilitare non solo il rapporto con la clientela e la gestione dei vari problemi informatici che ogni giorno possono venire a mostrarsi, ma è stato sviluppato con il vero scopo di aiutare i tecnici che lavorano ai problemi, e quindi vien data la possibilità di indicare metodo (Teleassistenza oppure On Side) e durata di un intervento, così da poter poi estrarre anche reportistiche utili a capire, ad esempio, il cliente più impegnativo, il cliente con meno problemi, e così via.
Altra particolarità di questo progetto (presente però anche in altri CRM) è l'implementazione di un lettore automatico di email, cioè un utente con problema non dovrà far altro che scrivere una email ad un indirizzo specificato, e tale lettore automatico aprirà un ticket automaticamente, oppure lo aggiornerà se la email richiama un ticket già aperto in precedenza.
In questo modo sia gli utenti che i tecnici non dovranno mai accedere al portale web del CRM per gestire un issue (o ticket), ma potranno farlo in maniera comodissima tramite le email.

L'applicativo è stato scritto tramite tecnologie .NET (ASPX nello specifico) con database SQL per la gestione dei dati.

Per la parte di front-end invece sono stati utilizzati:

- HTML, CSS, Javascript.
- IDE di sviluppo: Visual studio.
- PHP: usato per l'implementazione del lettore automatico di email.

3.9. CRM – O

Azienda	Advise SRL

CRM – O (Officina) è un software sviluppato per una delle più grandi concessionarie Peugeot e Volkswagen della Campania.

Il software nasce come esigenza alla seguente richiesta: le due grandi multinazionali nominate sopra, richiedono con cadenza mensile un file tracciato, in formato txt, contenente l'elenco dei clienti soddisfatti e non passati in officina durante il mese corrente.

La soddisfazione viene stabilita tramite un colloquio telefonico che viene effettuato.

Dunque, il software, aveva i seguenti compiti:

- Comunicare con i gestionali, ottenendo così i clienti da chiamare questo mese.
- Per rendere agile la chiamata, ossia senza l'onere di dover digitare il numero da chiamare, tale software comunicava con un sistema di chiamate **VoIP** (www.voispeed.com), in modo che l'operatore potesse effettuare chiamate direttamente dalla pagina web.
- Compilazione dei questionari, da effettuare, ovviamente, a chiamata conclusa col cliente (o durante).
- Generazione del file tracciato, rispettando le regole previste, ed invio di tale file alle case madri.

Applicativo scritto in .NET, e database SQL per l’archivio dei dati.

3.10. Movitel

Azienda	Advise SRL

Questo piccolo progetto nasce dalla completa integrazione con i sistemi Voispeed, azienda fornitrice di centralini professionali aziendali.

L'obiettivo di questa applicazione Desktop era quella di interfacciarsi con il sistema Voispeed in modo da replicare su schermo le funzionalità previste da un classico telefono centralino, esempio la possibilità di effettuare chiamate, inoltrare chiamate, ecc.

Le tecnologie utilizzate per implementare questo progetto sono:

- Microsoft .NET (VB.NET) come linguaggio di programmazione.
- HTML, CSS, Javascript per la parte front-end.
- SQL come database.
- API per comunicare con il centralino.

3.11. Acquisto Merci - Creazione Ordini

Azienda	Software Business SRL

Questa applicazione è stata implementata durante il mio tirocinio aziendale.

Essa nasce dall'esigenza dei clienti finali di ordinare merce in autonomia, e quindi l'applicativo, implementato in stile Master-detail, mostrava la lista di articoli acquistabili e la gestione di un carrello finale.

Dopodiché, ad ordine completo, esso veniva esportato nel sistema SAP di riferimento.

Le richieste verso il server SAP venivano eseguite tramite API REST.

Tecnologie

- HTML, CSS, Javascript per la parte front-end.
- XML e JSON per lo scambio di dati.
- API per creare l’ordine nel sistema cliente previsto.
- SAP WEB IDE come strumento di programmazione.

3.12. Freelance e Progetti personali

Nella mia esperienza ho anche lavorato su progetti al di fuori dell'azienda in cui ero impegnato.
Alcuni erano commissioni da clienti terzi, mentre altri erano idee personali che ho sviluppato autonomamente.

3.12.1. Quadro Luminoso – QL Mobile

Azienda	Ferrovie dello Stato Italiane

Progetto commissionato da Trenitalia, che aveva l'esigenza di monitorare, in maniera sicura e semplice, le varie tratte che il sistema ferroviario prevedeva.

Questo progetto nasce durante il periodo Covid, in cui, per sopperire all'impossibilità di lavorare in presenza in ufficio, venne ideata questa soluzione per permettere agli operatori di lavorare anche da casa.

Tutt'ora il sistema è in uso dalle stazioni di Bologna e Milano, ma col tempo la lista potrà solamente aumentare.

QL Mobile è una APP che, tramite sistemi di API, comunica con i sistemi RFI per ottenere informazioni riguardo le tratte, i ritardi, le videoregistrazioni, che un qualsiasi operatore poteva interrogare per qualsiasi esigenza.
Oltre ciò, la APP implementa anche un sistema di streaming video per accedere a varie telecamere poste su una specifica tratta.

Sono previste due modalità di visualizzazione:

Modalità Image: aggiornamento a intervalli regolari configurabili, che simula un frame rate (FPS) costante.

Modalità RTSP: riproduzione diretta di flussi video in tempo reale provenienti dal server.

Per ragioni di sicurezza, la APP non è pubblicata sul PlayStore ufficiale ed opera in maniera corretta solo in presenza di Rete mobile RFI sul device (SIM dedicata), cosi da controllare con facilità i dispositivi da cui la APP può funzionare.

L’APP viene distribuita attraverso uno store interno RFI, accessibile solo a personale autorizzato.
La mia responsabilità includeva il build locale del progetto e la pubblicazione interna sullo store RFI (la distribuzione finale era gestita da altri team).

La APP è stata sviluppata sia per Android che per iOS, scritta in entrambi casi con linguaggio nativo Android e Swift.

Strumenti coinvolti:

- Android (Java).
- Swift (IOS).
- API.

3.12.2. Fanta Regolamento

Sito web	https://paolinoangeletti.github.io/fantaregolamento

Questo progetto nasce da un gioco condiviso con amici: **il Fantacalcio**.

A partire dall’esigenza di definire un regolamento chiaro e automatizzato in grado di gestire le diverse situazioni che possono presentarsi, ho ideato e sviluppato una web application dedicata.

L’applicazione è stata deployata tramite **GitHub Pages** ed è stata realizzata utilizzando HTML, CSS e JavaScript.

In questo progetto ho anche avuto modo di realizzare Unit testing sfruttando **Jasmine**.

3.12.3. My PortFolio

Sito web	https://paolinoangeletti.github.io/MyPortFolio

Questa applicazione realizza un curriculum vitae navigabile, in cui ogni concetto è collegato a progetti, tecnologie, e viceversa.

L’applicazione è stata deployata tramite **GitHub Pages** ed è stata realizzata utilizzando il framework React, quindi HTML, CSS e Javascript XML.

Per il versionamento è stato utilizzato GIT.

4 Certificazioni

4.1. Corso sicurezza generale

ATTESTATO DI FREQUENZA E PROFITTO

Corso di Formazione Generale alla Salute e Sicurezza per i Lavoratori

Il corso di formazione è stato svolto in ottemperanza al D. Lgs. 81/08, art. 37, comma 7 e s.m.i e all'Accordo Stato Regioni del 21/12/2011

conferito a:

Angeletti Paolino

codice fiscale

NGLPLN93E04G812F

Ore di formazione previste	*Ore di formazione frequentate*
4	***4***

Principali argomenti trattati nel corso dell'azione formativa:

1. Concetti di rischio;
2. Danno;
3. Prevenzione;
4. Protezione;
5. Organizzazione della prevenzione aziendale
6. Diritti, doveri e sanzioni per i vari soggetti aziendali;
7. Organi di vigilanza, controllo e assistenza.

Soggetto erogatore: **Teoresi S.p.A.**

con sede a **Napoli** in **Via Ferrante Imparato, 198**

La formazione si è svolta dal **27/03/2024** al **27/03/2024**

Data **04/04/2024**

Il Datore di Lavoro
(timbro e firma)

Teoresi S.p.A.
via Perugia, 24 - 10152 Torino
c.f. e p.iva 03037960014

Il docente
Costabile Lo Schiavo

4.2. Corso sicurezza rischio basso

ATTESTATO DI FREQUENZA E PROFITTO

Corso di Formazione Specifica alla Salute e Sicurezza per i Lavoratori
RISCHIO BASSO

Il corso di formazione è stato svolto in ottemperanza al D. Lgs. 81/08, art. 37, comma 7 e s.m.i e all'Accordo Stato Regioni del 21/12/2011

conferito a:

Angeletti Paolino

codice fiscale

NGLPLN93E04G812F

Ore di formazione previste	*Ore di formazione frequentate*
4	***4***

Settore Ateco 2007: 62.02.00

Principali argomenti trattati nel corso dell'azione formativa:

1.	Rischi infortuni	2.	Stress lavoro correlato
3.	Meccanici generali	4.	Movimentazione manuale dei carichi
5.	Elettrici generali	6.	Segnaletica
7.	Emergenze	8.	Rischi specifici associati alla mansione
9.	Organizzazione del lavoro	10.	Procedure esodo e incendi
11.	Ambienti di lavoro	12.	Tutela della lavoratrice in gravidanza e allattamento

Soggetto erogatore: **Teoresi S.p.A**

con sede a **Napoli** in **Via Ferrante Imparato, 198**

La formazione si è svolta dal **27/03/2024** al **27/03/2024**

Data **04/04/2024**

Il Datore di Lavoro
(timbro e firma)

Teoresi S.p.A.
via Perugia, 24 - 10152 Torino
c.f. e p.iva 03037960014

Il docente
Costabile Lo Schiavo