UNA PANORAMICA COMPLETA DI MYKODE: UN NUOVO FRAMEWORK JAVASCRIPT RIVOLUZIONARIO PER LO SVILUPPO DI APPLICAZIONI WEB

Gaetano Lazzo, Antonio Piizzi, Donatello Vavallo, Francesco Capruzzi (
Tempo srl, vico Capurso 5 70126 BARI (BA), francesco.capruzzi@temposrl.com

SOMMARIO

Il *framework* JavaScript denominato MyKode è utile allo sviluppo di applicazioni web che dispone di una struttura unica e di una modellazione funzionale, capaci di affrontare le sfide contemporanee del settore. Il cuore del *framework* è la gestione avanzata dei dati che utilizza dati strutturati in maniera gerarchica nella cache del *browser*, potenziata da librerie come jsDataSet e jsDataQuery. Il suo modello di *binding* avanzato collega i controlli dell'interfaccia utente ai dati tramite attributi HTML5, semplificando la progettazione delle pagine web. Un'esclusiva di MyKode è la gestione lato client di operazioni CRUD su dati eterogenei, che consente il salvataggio dei dati multi-tabella in un'unica transazione. Il suo sistema di gestione automatizzato per le barre degli strumenti e le interazioni di controllo centralizzate riduce i tempi di sviluppo. La struttura modulare del *framework* è compatibile con due *back-end* specifici, garantendo versatilità. Con la sua compatibilità multipiattaforma, MyKode si propone di mutare significativamente lo sviluppo di applicazioni web contando su maggiore efficienza e flessibilità.

INTRODUZIONE

Nell'era digitale, lo sviluppo di interfacce utente per applicazioni web richiede un'attenzione costante alle evoluzioni tecnologiche e alle innovazioni nel campo della programmazione web. Per lo sviluppo web, l'adozione di *framework* è fondamentale per gli sviluppatori che aspirano a creare interfacce utente moderne, efficienti e scalabili. Attualmente esiste un'ampia gamma di *framework*, ognuno con caratteristiche uniche, progettati per semplificare lo sviluppo e migliorare l'efficienza del codice. La scelta del miglior *framework* determina il successo di un progetto. Esistono molti *framework front-end* popolari che hanno semplificato notevolmente l'intero processo di sviluppo web, tra cui React, Angular e Vue. Mentre, dagli ambienti runtime Javascript ai framework lato server, i *framework* JS che abilitano il livello *back-end* includono Node.js, Express e Meteor.js (Alok Sinha Ranjan, 2020).

In questo contesto, gli sviluppatori sono impegnati a selezionare la soluzione ideale tra quelle disponibili per tradurre le idee in interfacce utente ricche di funzionalità e adattabili.

L'analisi dei linguaggi di programmazione web per l'implementazione di *framework* richiede anche una valutazione approfondita delle tendenze del settore e delle esigenze specifiche del progetto. Questo articolo presenta un *framework* innovativo, chiamato MyKode, che apporta notevoli progressi nel campo dello sviluppo di applicazioni web. Esso è open source ed è disponibile su GitHub. A differenza di quelli esistenti, MyKode offre ai programmatori una struttura ben definita e riutilizzabile per i moduli software, insieme a una suite di logiche di business pre-implementate, interazioni per i controlli grafici e comunicazioni con il server web. Questo *framework* impiega una struttura dati gerarchica nella *cache* di memoria del *browser*, imitando l'architettura delle tabelle di database remote e le loro relazioni, facilitata da librerie JavaScript specializzate come jsDataSet e jsDataQuery. Inoltre, introduce un modello di *binding* avanzato tra i controlli dell'interfaccia utente e i dati sottostanti utilizzando attributi dati HTML5 (Lawson e Sharp, 2011), semplificando il processo di progettazione delle pagine web. Un altro aspetto degno di nota è la sua gestione lato client di operazioni CRUD su dati eterogenei, che consente il salvataggio di dati da diverse tabelle in un'unica transazione, una capacità solitamente non presente nei principali *framework front-end*.

MyKode automatizza anche la gestione delle barre degli strumenti e delle azioni associate, riducendo così i tempi di sviluppo attraverso la generazione automatica di *query* e il popolamento dei dati nell'interfaccia grafica. Inoltre, centralizza le interazioni tra controlli complessi, come quelli visibili nelle relazioni *master-detail* e nelle dipendenze nelle griglie o nelle viste ad albero.

La sua struttura modulare, che supporta l'estensione delle funzionalità, offre agli sviluppatori la flessibilità di integrare la propria logica di business senza soluzione di continuità. Inoltre, il *framework* garantisce la completa interoperabilità con diversi *back-end* (ad esempio 'myKode_core'). La combinazione di queste caratteristiche segna un cambiamento nello sviluppo di applicazioni web, caratterizzato da un design incentrato sull'utente, compatibilità multipiattaforma (Perakakis e Ghinea, 2015) e una maggiore efficienza di sviluppo.

Lo sviluppo della soluzione JavaScript per la creazione di applicazioni mira a soddisfare la crescente domanda di mercato di applicazioni multipiattaforma. Ci si aspetta sempre più che queste applicazioni siano compatibili con diversi dispositivi desktop e mobili e *browser*. Questa tendenza sottolinea la necessità di un *framework* che sia non solo versatile, ma anche intuitivo e accattivante in

termini di esperienza grafica.

La filosofia alla base della soluzione presentata in questo articolo, denominata MyKode, è, quindi, duplice:

- mira a fornire ai programmatori gli strumenti per sviluppare rapidamente applicazioni web complesse e di facile manutenzione;
- si impegna a garantire un'esperienza di lavoro fluida, chiara e visivamente accattivante.

Per raggiungere questi obiettivi, l'utilizzo di librerie open source consolidate come "Jquery", "lodash" (Kholmatov, 2023) per le funzioni di utilità JavaScript e "Bootstrap" per i layout grafici responsive (Spurlock, 2013) è stato massimizzato, riducendo i costi e garantendo un lungo ciclo di vita al software.

SVILUPPO E ARCHITETTURA DEL *FRAMEWORK*

La creazione di un'architettura software robusta, scalabile e manutenibile necessita di una combinazione di principi di modellazione funzionale e strutturale. Questa, basata sul concetto di funzioni pure e immutabilità dei dati, enfatizza vantaggi come:

- maggiore sicurezza del codice,
- facilità di test
- migliore riutilizzo del codice.

Questo approccio è stato combinato con i linguaggi di modellazione web (De Bruijn et al., 2006), tra cui HTML, CSS, JavaScript, TypeScript, JSON, XML, GraphQL e YAML, per modellare le interfacce web e le funzionalità avanzate delle moderne applicazioni web.

In generale, la modellazione strutturale delle applicazioni web è guidata dai principi chiave di progettazione del software e dai principi SOLID (Marcotte, 2020), che insieme mirano a migliorare la qualità del codice e semplificare la gestione del progetto.

I principi di progettazione (Hung, 2001) includono la Separation of Concerns (SoC), che sostiene la suddivisione di un sistema complesso in componenti distinti, ciascuno responsabile di una funzionalità specifica. Questa suddivisione migliora la manutenibilità, l'estensibilità e la comprensibilità del codice. Il principio "Keep It Simple Stupid" (KISS) sottolinea l'importanza della semplicità nella progettazione e nei progetti di sistema, promuovendo soluzioni intuitive e dirette. Il principio "Don't Repeat Yourself" (DRY) si concentra sulla riduzione della duplicazione del codice, incoraggiando l'astrazione e il riutilizzo di parti di codice comuni, rendendo così il codice più manutenibile, chiaro ed efficiente.

A complemento di questi principi c'è il principio "*You Aren't Gonna Need It*" (YAGNI), che suggerisce di evitare funzionalità o complessità non necessarie finché non siano assolutamente essenziali, risparmiando così tempo e riducendo potenziali bug.

Il Test-Driven Development (TDD) è impiegato come pratica di sviluppo fondamentale, seguendo un ciclo iterativo di scrittura di test automatizzati, implementazione del codice minimo necessario e refactoring. Questo approccio garantisce una copertura completa dei test, mantenendo la pulizia e la struttura del codice nel tempo e facilitando lo sviluppo incrementale.

Il principio di Inversione del Controllo (IoC), che include Dependency Injection e Service Locator, svolge un ruolo cruciale nel disaccoppiamento di classi e componenti, rendendo il sistema più modulare e facilitando l'introduzione di modifiche e nuove funzionalità senza intervenire sulle classi esistenti.

Inoltre, viene adottato il principio di Convenzione sulla Configurazione (CoC), che riduce la necessità di configurazioni esplicite seguendo convenzioni predefinite. Questo principio aumenta la produttività degli sviluppatori, consentendo loro di concentrarsi maggiormente sulla logica di business e sulle funzionalità dell'applicazione piuttosto che sui dettagli di basso livello. Insieme, questi principi costituiscono il fondamento dell'architettura del *framework* JavaScript, garantendone l'adattabilità, la robustezza e la semplicità d'uso per gli sviluppatori, rispondendo al contempo alle mutevoli esigenze dello sviluppo di applicazioni web moderne.

L'architettura software è la struttura fondamentale che determina la progettazione e l'organizzazione interna di un sistema, fornendo la base su cui costruire soluzioni applicative specifiche. Un sistema ben progettato offre indicazioni chiare agli sviluppatori, promuovendo coerenza, riutilizzabilità del codice e facilità di estensione.

In questo contesto, l'architettura diventa cruciale poiché influenza direttamente l'esperienza degli sviluppatori e la capacità dell'applicazione di adattarsi a diverse esigenze e scenari. Un'architettura robusta deve bilanciare flessibilità e coerenza, fornendo un ambiente in cui gli sviluppatori possano concentrarsi sulla logica specifica dell'applicazione piuttosto che sulle decisioni relative all'infrastruttura.

Un *framework* ben progettato (è il caso di MyKode) permette la suddivisione delle funzionalità in moduli distinti e interconnessi. Questa modularità facilita la comprensione del sistema e consente l'estensione o la sostituzione di parti specifiche senza compromettere l'intero *framework*.

La definizione di interfacce (API) chiare e coerenti è fondamentale perché facilita l'interazione degli sviluppatori con il *framework*, consentendo loro di utilizzare ed estendere le funzionalità in modo intuitivo.

MyKode è stato progettato per garantire una sua facile estensione e configurazione. Gli sviluppatori potranno adattare il comportamento del *framework* alle esigenze specifiche dell'applicazione senza dover apportare modifiche sostanziali al codice sorgente. Inoltre, l'architettura è stata progettata tenendo conto della scalabilità, garantendo di poter gestire complessità e volumi di dati crescenti senza compromettere le prestazioni.

Il *framework* MyKode è stato pensato come un *rich-client* e come una *Single Page Application* (SPA): il termine *rich-client* si riferisce a un'applicazione software che presenta un elevato numero di funzionalità e complessità eseguite sul lato client.

Un *rich-client* offre solitamente un'interfaccia utente ricca e interattiva (Dissanayake, Liyanage e Dias, 2015). Questa può includere animazioni, transizioni fluide e un aspetto sofisticato. Una delle sue peculiarità è la possibilità di conservare una certa quantità di dati localmente, consentendo all'applicazione di funzionare offline o di rispondere alle richieste degli utenti senza una connessione costante al server. Un'applicazione di questo tipo può accedere alle risorse locali del dispositivo, come la fotocamera, il GPS o la memoria locale, per offrire funzionalità avanzate e personalizzate.

Le SPA rappresentano un paradigma di sviluppo web che ha radicalmente trasformato l'esperienza utente online. A differenza delle tradizionali applicazioni web multipagina, esse adottano un approccio innovativo incentrato su una singola pagina dinamica, consentendo un'interazione fluida e fluida per gli utenti.

Per creare il *framework* MyKode, sono stati creati: il dataset e i MetaData per il *back-end*, la pagina HTML per la gestione del *front-end* e i file JavaScript per le pagine MetaPage e per i MetaData. Nella fase di creazione del dataset, è stato creato lo schema logico e sono state identificate le relazioni tra le tabelle.

MetaData è la classe che implementa i metodi standard autorizzati a implementare determinate funzioni del *framework*. Il compito del framework MyKode è di garantire che ogni metodo implementato venga chiamato al momento necessario.

La progettazione del *framework front-end* si è concentrata sulla presenza di un oggetto MetaApp che gestisce le varie pagine web; infatti, l'applicazione sviluppata con MyKode presenta molteplici pagine web dinamiche e interattive e per questo motivo è stata progettata la pagina HTML statica in cui sono contenuti i vari attributi-dati. Per ogni pagina web è stato creato un file javascript che ne gestisce la logica e l'interazione con il framework. Ogni MetaPage derivata può sovrascrivere i metodi della classe base per ottenere un comportamento personalizzato, oppure implementare metodi da inserire nel ciclo di vita della Page.

Ogni MetaPage è collegata al suo file HTML che ne rappresenta la vista, tramite il meccanismo di *binding*. Attraverso il meccanismo di derivazione tra pagine, è possibile centralizzare la logica comune tra una o più applicazioni all'interno di un singolo file.

Il *front-end* di myKode è un *framework* per lo sviluppo di moduli web con codice minimo. Utilizza un *front-end* JavaScript per un'interazione avanzata con i client senza necessità di *post-back* continui. Il *back-end*, disponibile sia in versione Node.js che .NET, facilita le operazioni sui moduli web, tra cui ricerca, modifica, inserimento e apertura dei dettagli del modulo. I *form* sono associati a un DataSet, che rappresenta una copia locale delle righe del database. La tabella principale (entità) può avere sottoentità (dettagli), collegate tramite campi chiave specifici. Il *framework* garantisce connessioni logiche tra entità e sottoentità, gestendo efficacemente le operazioni sui dati. Il *framework* consente la lettura e la scrittura automatica del DataSet, caricandolo quando un utente seleziona una riga e gestendo anche le eliminazioni. La visualizzazione dei dati è progettata con tag HTML, con campi collegati alle tabelle utilizzando il formato "table.field". I comportamenti possono essere aggiunti ai *form* senza manipolare direttamente i controlli HTML, favorendo l'indipendenza del codice. La struttura dell'applicazione prevede l'associazione di ciascuna tabella a una classe MetaData e la creazione di pagine con classi MetaPage. La classe MetaApp gestisce il livello più alto dell'applicazione, gestendo pagine e metadati. MetaData descrive le proprietà delle tabelle, evitando ripetizioni di codice e facilitando la manutenzione. MetaPage contiene codice comune per le pagine, offrendo *hook* per funzioni specifiche. La struttura del progetto sul lato *front-end* ha un proprio file di configurazione in cui vengono inserite le impostazioni di base del progetto e le informazioni sulle librerie con le versioni specifiche incluse e utilizzate.

Il *back-end* myKode è un *framework* Node.js progettato per semplificare lo sviluppo di applicazioni Node.js. Fornisce classi per l'accesso a database relazionali, utilizzando jsDataSet per gestire set di dati in memoria in modo simile ai DataSet di .NET . Il *framework* enfatizza l'accesso ai dati, la logica di business, la sicurezza e la creazione di DataSet

L'autenticazione è gestita tramite JSON Web Token (JWT), con jsApplication che gestisce le sessioni utente e i percorsi di accesso. L'accesso ai dati prevede tre classi principali:

- DataAccess per le operazioni di basso livello,
- GetData per la lettura di DataSet
- PostData per la scrittura di DataSet.

Le regole di business sono implementate tramite jsBusinessLogic e i controlli di sicurezza sono gestiti da jsSecurity .

Il *framework* supporta la creazione di DataSet tramite metodi manuali, metodi semi-manuali utilizzando DbDescriptor o la creazione visiva con lo strumento HDSGene.

Le classi MetaData centralizzano le informazioni delle tabelle, tra cui la validità dei dati di riga, i valori predefiniti, gli schemi di campo ad incremento automatico e le strutture degli elenchi.

Allora, MyKode *back-end* semplifica lo sviluppo di applicazioni Node.js fornendo classi per l'accesso al database, l'autenticazione, l'accesso ai dati, la logica aziendale, la sicurezza, la creazione di DataSet e informazioni centralizzate sulle tabelle tramite classi MetaData.

INNOVAZIONI E SOLUZIONI SVILUPPATE

Il *framework* MyKode offre una soluzione versatile e ad alte prestazioni che bilancia sapientemente innovazione e praticità, risolvendo sfide nuove. Lo sviluppo del *framework* JavaScript integra una moltitudine di *pattern* di progettazione innovativi e principi di ingegneria del software, dando vita a un *framework* efficiente e adattabile per lo sviluppo di applicazioni web. Uno dei *pattern* architetturali impiegati è il Model-View-Controller (MVC), che organizza il codice in tre componenti distinti:

- Model;
- View;
- Controller.

Questa struttura migliora la modularità e la manutenibilità del codice e ne facilita la scalabilità. Il Model gestisce la logica di business e i dati, la View gestisce la presentazione dei dati e il Controller funge da intermediario, elaborando gli input dell'utente e aggiornando di conseguenza Model e View.

La separazione semplifica i test e il riutilizzo del codice, migliorando significativamente la robustezza e l'efficienza complessive del *framework*. L'Observer Pattern, poi, è un *pattern* di progettazione comportamentale che consente una comunicazione efficiente e disaccoppiata tra gli oggetti del sistema, gestendo la dipendenza uno-a-molti tra gli oggetti e consentendo la notifica automatica dei cambiamenti di stato a tutti gli osservatori interessati. Esso favorisce flessibilità ed estensibilità del sistema, soprattutto in ambienti *multi-thread*, migliorando l'efficienza e la gestibilità della gestione degli eventi.

Il Factory Method Pattern, parte integrante del *framework*, fornisce un'interfaccia per la creazione di istanze di classe, lasciando la scelta dell'istanziazione concreta delle classi alle sottoclassi.

Questo approccio, basato sul principio dell'*Inversion of Control*, consente una maggiore flessibilità nella creazione di oggetti e riduce l'accoppiamento tra codice client e classi concrete. Il *pattern* contribuisce a mantenere una base di codice pulita e gestibile, soprattutto nei sistemi in cui la logica di creazione degli oggetti varia significativamente. Inoltre, il Composite Pattern viene utilizzato per gestire in modo uniforme singoli oggetti e composizioni, creando strutture di oggetti ad albero. Esso consente la composizione ricorsiva e tratta tutti gli oggetti tramite un'interfaccia comune, semplificando le interazioni del client con il sistema.

Questi *design pattern*, insieme alle tecnologie web avanzate utilizzate (HTML5, CSS3, JavaScript, WebAssembly, GraphQL, PWA, architetture *serverless* e strumenti di sviluppo come Webpack, Babel e npm), contribuiscono collettivamente alle funzionalità e alle nuove soluzioni parte integrante del *framework*. Essi rispondono alle esigenze critiche delle moderne applicazioni web:

- alte prestazioni,
- interfacce intuitive e
- gestione efficace delle interazioni e delle visualizzazioni di dati complesse.

Il carattere innovativo di questo *framework* risiede nel fornire una struttura chiara e riutilizzabile di moduli software e una suite di logiche di business pre-implementate, interazioni tra controlli grafici e interazioni con il server web. Pertanto un elenco delle principali innovazioni del *framework* può rappresentarsi come:

1. una struttura dati gerarchica nella cache di memoria del browser, rappresentata virtualmente come la struttura delle tabelle di database remoti, le loro proprietà e relazioni. Questo risultato è ottenuto tramite librerie JavaScript appositamente sviluppate, come jsDataSet e jsDataQuery , disponibili in repository pubblici per l'analisi da parte della comunità open source.
2. un modello di associazione avanzato tra controlli e dati sottostanti, facilitato dagli attributi dati HTML5, che consente ai programmatori di configurare le informazioni di associazione durante la progettazione della pagina web.
3. una gestione lato client di operazioni CRUD su dati eterogenei, che consente il salvataggio nel database di informazioni provenienti da tabelle diverse in un'unica transazione, funzionalità rara nei principali *framework front-end*.
4. una gestione automatizzata delle barre degli strumenti e delle azioni associate, con notevole risparmio di tempo di sviluppo grazie all'automazione della generazione di *query* e al popolamento dell'interfaccia grafica con i dati richiesti.

5. una centralizzazione delle interazioni ricorrenti tra controlli complessi, come relazioni master-dettaglio e dipendenze tra controlli quali griglie o visualizzazioni ad albero.
6. una struttura modulare con la possibilità di estendere le funzionalità, a garanzia di flessibilità nell'incorporare particolari logiche, senza vincoli.
7 una progettazione incentrata sull'utente, sulla compatibilità multipiattaforma e su di una maggiore efficienza di sviluppo.

DISCUSSIONE E CONCLUSIONI

La concezione di MyKode affonda le radici nella necessità di colmare le lacune esistenti, offrendo una soluzione in grado di soddisfare nuove e dinamiche esigenze di sviluppo web in continua evoluzione. Si tratta di un'innovativa testimonianza della potenza dell'integrazione, in particolare nella gestione dati. L'integrazione di una struttura dati gerarchica direttamente nella cache di memoria del browser, tramite l'utilizzo di librerie JavaScript specializzate come jsDataSet e jsDataQuery, è una soluzione strategica. Essa aumenta le prestazioni e rimodella la comprensione della gestione dati complessi all'interno delle applicazioni web.

Per quanto concerne la semplificazione del processo di progettazione, l'utilizzo da parte del *framework* di attributi dati HTML5 per migliorare i modelli di *binding* si scosta decisamente dalle metodologie tradizionali, fornendo agli sviluppatori strumenti più efficienti e intuitivi, accorciando efficacemente il ciclo di sviluppo e semplificando la progettazione delle applicazioni.

Una caratteristica da sottolineare è la gestire DI operazioni CRUD su dati eterogenei lato client. Ciò consente il salvataggio simultaneo di dati da tabelle diverse in un'unica transazione, funzione per nulla diffusa nelle soluzioni comunemente disponibili. Ciò rappresenta una chiara indicazione della robustezza del *framework* e della sua adattabilità nella gestione di complesse interazioni di dati, cruciali nello sviluppo di applicazioni web moderne.

Il *framework* innova anche nel modo in cui automatizza la gestione delle barre degli strumenti e centralizza le interazioni tra controlli complessi, come le relazioni *master-detail*. Tali innovazioni contribuiscono a un processo di sviluppo più snello, alleggerendo il carico di lavoro legato alla creazione di applicazioni web sofisticate.

MyKode anticipa, poi, le tendenze e le esigenze future nello sviluppo di applicazioni web. Con la crescente domanda di applicazioni multipiattaforma, intuitive ed esteticamente gradevoli, un *framework* deve essere potente e adattabile.

La struttura modulare del framework MyKode e la sua compatibilità con diversi sistemi *back-end* ne evidenziano la versatilità, rendendolo adatto a diversi scenari di sviluppo. Inoltre, l'attenzione del *framework* al design incentrato sull'utente e alla compatibilità multipiattaforma è in linea con le mutevoli aspettative di utenti finali e sviluppatori.

Combinando questi elementi con una maggiore efficienza nello sviluppo, MyKode si candida a nuovo punto di riferimento per lo sviluppo di applicazioni web, ben bilanciando funzionalità ed esperienza utente.

ACKNOWLEDGMENTS

Questa ricerca è stata finanziata da "Fondo Europeo di Sviluppo Regionale Puglia POR Puglia 2014 – 2020 "Investiamo nel vostro futuro" – Asse prioritario I obiettivo specifico 1a Azione 1.1 Sub -azione 1.1.a" - "PROGETTO EOLO - VEICOLO PER NUOVI CONTENUTI" - Codice Progetto M95VIF6.

BIBLIOGRAFIA

Alok Sinha Ranjan, ABR (2020) *JavaScript per lo sviluppo web moderno*. BPB Publications. Disponibile all'indirizzo: https://books.google.it/books?id=5D4WzgEACAAJ

De Bruijn, J. et al. (2006) *Il linguaggio di modellazione dei servizi web WSML: una panoramica*, in Conferenza europea sul web semantico . Springer, pp. 590–604.

Disney, MS (2006) *Un confronto delle prestazioni dei server http Apache e IIS*, inviato all'Oslo University College, MS009A [Preprint].

Dissanayake, NR, Liyanage, U. e Dias, K. (2015) *Un concetto di client bilanciato per applicazioni internet ricche*, in International Research Symposium on Engineering Advancements (RSEA 2015) .

Hung, D. (2001) *Principi di progettazione per l'apprendimento basato sul web: implicazioni dal pensiero vygotskiano*, Educational Technology , 41(3), pp. 33–41.

Kholmatov, A. (2023) *Librerie ampiamente utilizzate nel linguaggio di programmazione javascript e le loro capacità*, Intent Research Scientific Journal , 2(10), pp. 18–25.

Lawson, B. e Sharp, R. (2011) *Introduzione a HTML5 . New Riders (Voci che contano)*.Disponibile all'indirizzo: https://books.google.it/books?id=a2BVmAEACAAJ

Marcotte, C.-H. (2020) *Una guida atipica ai modelli di progettazione ASP.NET Core 5: un'avventura SOLIDA nei principi architettonici, nei modelli di progettazione*, in .NET 5 e in C. Packt Publishing Ltd.

Perakakis, E. e Ghinea, G. (2015) *Tecnologie HTML5 per una pubblicità TV interattiva/intelligente multipiattaforma efficace*, IEEE Transactions on human-machine systems , 45(4), pp. 534–539.

Spurlock, J. (2013) *Bootstrap: sviluppo web reattivo*, O'Reilly Media, Inc.

TempoSrl/myKode_core (2022). TempoSrl. Disponibile all'indirizzo: https://github.com/TempoSrl/myKode_core

TempoSrl / myKode_Backend ' (2022). Tempo Srl . Disponibile all'indirizzo: https://github.com/TempoSrl/myKode_Backend

TempoSrl / myKode_Frontend ' (2022). Tempo Srl . Disponibile all'indirizzo: https://github.com/TempoSrl/myKode_Frontend