MP

MIGRATION

HardShipV2

Versione 1.0

Sommario

Introduzione 3

Analisi Struttura Esistente 3

Valutazione Tecnologie Disponibili 5

- **Server** 5
- **Client** 5

Tecnologie per l'implementazione 5

- **Server** 5
- **API REST** 6
- **I vantaggi di REST per lo sviluppo** 6
- **Client** 6

Tecnologie scelte per l'implementazione 7

Nuova Struttura 10

Rischi del nuovo sistema 12

Requisiti del sistema 12

Riutilizzo delle componenti 13

Testing 14

Definizione Strategia 14

Fattibilità della Strategia 15

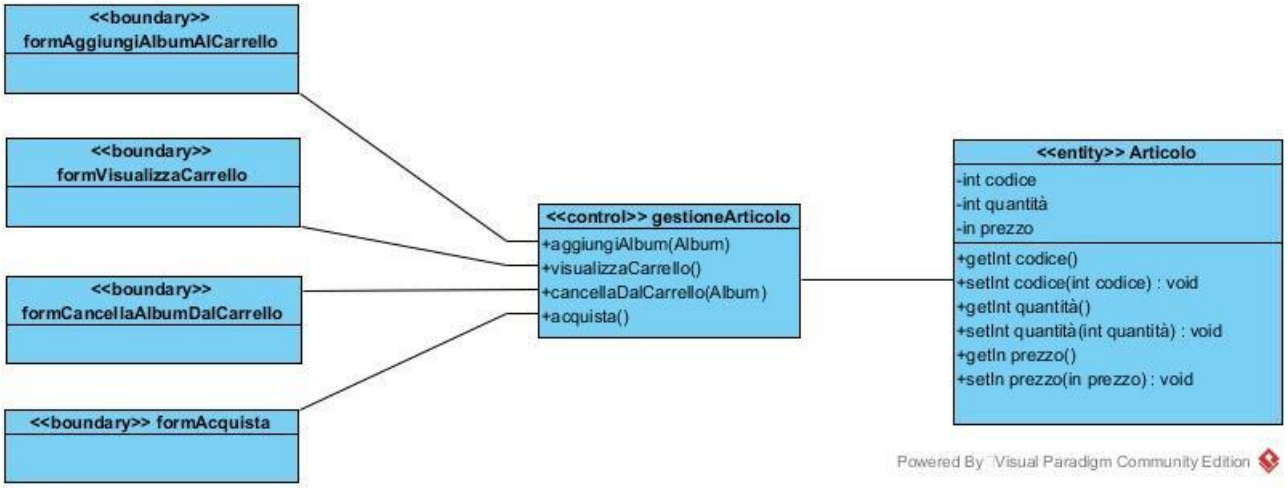
Determinare le risorse necessarie 15

1.Introduzione

In questo documento verranno descritte le motivazioni e le scelte da noi effettuate per la migrazione del vecchio sistema. Nella prima fase è stata effettuata l'analisi del sistema per verificare la conformità dei documenti con la progettazione e l'implementazione e non solo anche per acquisire informazioni sul sistema per non ripetere gli stessi errori. Un'altra parte fondamentale che abbiamo riscontrato dall'analisi è l'assenza di test, parte fondamentale per un software. Prima di effettuare la migrazione verso il nuovo sistema siamo andati ad analizzare le tecnologie, architetture e design a nostra disposizione per avere un confronto di quale fosse la migliore scelta per il nuovo sistema e che rispecchiasse i nuovi requisiti del business.

2.Analisi Struttura Esistente

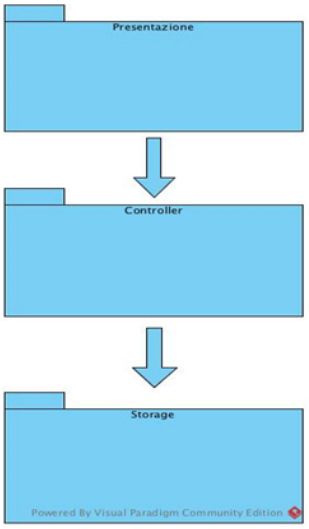
In questa fase è stato analizzato il sistema corrente da diversi punti di vista, in primo luogo si è partiti dalla documentazione per controllare i Requisiti Funzionali e non Funzionali cercando di comprendere la progettazione e l'implementazione del software esistente.



Nella figura 1.1 è riportato la questione descritta precedentemente quindi l'assenza di un pattern strutturale dello storage cioè il DAO

Ciò ha portato all'individuazione del principale problema del software ovvero la mancanza di un design pattern strutturale che gestisca il Database, in terzo luogo non sono stati individuati casi di test per poter effettuare test d'unità.
L'architettura utilizzata nell'implementazione della piattaforma è quella Client-Server, un server che gestisce tutto ciò che riguarda i dati persistenti e l'accessibilità da parte degli utenti mentre il client ha il compito di presentare i contenuti agli utenti finali. Il sistema basandosi sulla documentazione è diviso in 3 layer: **Model** (gestione dati persistenti), **View** (gestione interfaccia grafica), **Controller** (gestione logica di sistema); i tre layer sono organizzati secondo il noto pattern 'Model-View-Controller'. Per la progettazione e lo sviluppo di HardshipV1 è stato scelto il design pattern MVC e utilizzando il metodo del

forward engineering è stato possibile analizzare il codice da cui si evince che ciò che è stato descritto nella documentazione non fosse rispettato nella fase di implementazione e progettazione, in quanto non vi è l'esistenza di un design pattern che gestisca lo storage e non vi è traccia del layer Storage. La seguente è un esempio di Servlet (Visualizza News) che come si può notare presenta le chiamate al Database direttamente senza utilizzare un design pattern e fa notare l'assenza del layer.



Nella figura 1.2 è riportato il modello MVC

Quindi non è presente l'oggetto Data Access Object (DAO) che serve a gestire la connessione con l'origine dati per ottenere e archiviare i dati. Astrae l'implementazione di accesso ai dati sottostante per il Business Object per consentire l'accesso trasparente all'origine dati. Un'origine dati potrebbe essere qualsiasi database come RDBMS, repository XML o file system flat ecc quindi vi è presente un alto accoppiamento.

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
 *      response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Connection conn = null;
    String driver = "com.mysql.jdbc.Driver";
    Statement st;
    try {
        Class.forName(driver).newInstance();
        conn = DriverManagerConnectionPool.getConnection();
        String id = request.getParameter("news");
        News b = new News();
        st = conn.createStatement();
        String query = "SELECT * FROM news WHERE ID=" + id + "";
        ResultSet rs = st.executeQuery(query);
        while (rs.next()) {
            b.setId(rs.getInt(1));
            b.setContenuto(rs.getString(2));
            b.setData(rs.getString(3));
            b.setAutore(rs.getString(4));
            b.setTitolo(rs.getString(5));
            b.setCopertina(rs.getString(6));
            b.setCategoria(rs.getString(7));
        }
        request.setAttribute("news", b);
        response.sendRedirect("pages/readmore.jsp?cod=1");
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Nella figura 1.3 è riportato la questione descritta precedentemente quindi l'assenza di una parte dello storage cioè il DAO

3. Valutazione Tecnologie Disponibili

- ### Server

  Sono state analizzate diverse opzioni per poter scegliere quella ritenuta migliore per il nostro caso, sotto l'elenco delle possibili opzioni.

  - Tecnologia REST API con Spring Boot
  - REST API con NODEJS
  - Modifica delle Servlet per creare un modello MVC
  - Microservice

- ### Client

  Sono state analizzate diverse opzioni per poter scegliere quella ritenuta migliore per il nostro caso, sotto l'elenco delle possibili opzioni.

  - Flutter
  - Angular
  - HTML
  - Non toccare le JSP

4. Tecnologie per l'implementazione

- ### Server

Inizialmente è stato scelto di modificare le Servlet per correggere il sistema, poi considerando l'analisi effettuata tramite il forward engineering che evince la mancanza di alcune componenti, di una architettura effettiva e il testing, è stato pensato che fosse troppo dispendioso in termini di tempi e di costi riprendere le Servlet e quindi di conseguenza ci siamo affidati ad una tecnologia a noi più familiare ovvero REST API.

Un'ulteriore motivazione è che le tecnologie utilizzate risultano essere obsolete e poco scalabili. Dove la scalabilità risulta essere un nuovo requisito non funzionale da parte del nostro business, e grazie alla scelta dell'assenza di stato fornita dalle REST API tutto questo risulta essere più semplice.

Con la nostra scelta ci sono state delle conseguenze. Praticamente con l'implementazione Rest le interazioni tra client e server sono prive di stato, questo significa che il server non deve tenere traccia dello stato del client in memoria pertanto tutte le informazioni necessarie al server per restituire i dati devono essere incluse in ogni richiesta. L'assenza di stato trasferisce la responsabilità di mantenere lo stato al lato client così il server diventa più leggero in quanto non deve memorizzare più le stesse informazioni. Di seguito una descrizione della tecnologia scelta ed i suoi vantaggi.

- ### API REST

Un'**API REST**, nota anche come API RESTful, è un'interfaccia di programmazione delle applicazioni (API o API web) conforme ai vincoli dello stile architetturale REST, che consente l'interazione con servizi web RESTful. Con il termine REST intendiamo REpresentational State Transfer. Esso permette a tutti gli effetti di interfacciare un utente di un sistema alle risorse o servizi web offerti dal sistema.

- ### I vantaggi di REST per lo sviluppo

1. Separazione tra client e server: il protocollo REST separa totalmente l'interfaccia utente dal server e dall'archiviazione dei dati. Questo ha alcuni vantaggi quando si fanno sviluppi. Ad esempio, migliora la portabilità dell'interfaccia su altri tipi di piattaforme, aumenta la scalabilità dei progetti e consente di evolvere autonomamente le diverse componenti degli sviluppi.

2. Visibilità, affidabilità e scalabilità. La separazione tra client e server ha un evidente vantaggio, e cioè che ogni team di sviluppo può scalare il prodotto senza troppi problemi. Possono migrare ad altri server o apportare tutti i tipi di modifiche nel database, a condizione che i dati di ogni richiesta vengano inviati correttamente. La separazione rende più facile avere la parte anteriore e posteriore su server diversi e questo rende le app più flessibili con cui lavorare.

3. L'API REST è sempre indipendente dal tipo di piattaforma o linguaggio: l'API REST si adatta sempre al tipo di sintassi o piattaforme utilizzate, il che offre una notevole libertà quando si modificano o si testano nuovi ambienti all'interno dello sviluppo. Con un'API REST puoi avere server PHP, Java, Python o Node.js. L'unica cosa è che è indispensabile che le risposte alle richieste avvengano sempre nel linguaggio utilizzato per lo scambio di informazioni, normalmente XML o JSON.

- ### Client

Inizialmente è stato scelto di non toccare le JSP in modo da riutilizzare almeno il Presentation Layer, ciò non è stato possibile in quanto è subentrato un requisito non funzionale ovvero la scalabilità, ciò che si voleva era un sistema multipiattaforma , quindi mobile, web, e dispositivi desktop(windows e mac os). Quindi siamo andati a confrontare le due tecnologie e capire cosa potessero offrire per le nostre esigenze.

5.Tecnologie scelte per l'implementazione

La scelta è ricaduta su questi 3 framework Angular , React Native e Flutter. Quindi ciò che è stato fatto è mettere a confronto questi tre framework di sviluppo mobile multipiattaforma più popolari che vengono utilizzati per creare migliaia di app mobile.
Per la maggior parte del decennio, Angular è stato sovrano. D'altra parte sia React Native che Flutter sono relativamente nuovi e presentano molti vantaggi simili, strumenti e funzionalità native. Sono stati confrontati React Native, Flutter e ANgular in base a diversi fattori come:

- **Lingua**
- **Prestazioni**
- **Interfaccia utente (UI / UX)**
- **Supporto della community**

LINGUA

- **React Native:** si basa sul linguaggio JavaScript ES6 insieme a JSX che è un'estensione della sintassi di Javascript che rende il codice Javascript mirror scritto in HTML. Gli sviluppatori trovano facile la codifica in Javascript e a sua volta anche il suo apprendimento.
- **Flutter:** Le app Flutter sono scritte in linguaggio Dart che è relativamente facile da imparare e capire ed è un linguaggio di programmazione promettente per lo sviluppo multipiattaforma. Tuttavia non ha la stessa popolarità di Javascript. Uno sviluppatore che prova flutter per la prima volta dovrebbe imparare prima Dart. In genere gli sviluppatori che provengono però da un background C++ / Java possono apprendere Dart in maniera semplice e rapida perché gode delle stesse caratteristiche di un linguaggio di programmazione ad oggetti.
- **Angular:** Utilizza TypeScript, che è un superset di JS creato per progetti più estesi. TypeScript è relativamente compatto rispetto a JavaScript. Ciò facilita la navigazione e rende il processo di refactoring del codice più semplice e veloce. Angular è una riscrittura di AngularJS, un framework JavaScript, che è stato migliorato per un utilizzo TypeScript più user-friendly.

Prestazioni

- **React Native:** Idealmente si necessita di un bridge in React per chiamare le API Swift, Windows, Android o Mac. Gli sviluppatori devono affrontare problemi durante la creazione di app ibride, ma raramente incontrano problemi relativi alle prestazioni per le app native. React offre prestazioni senza soluzione di continuità in tutti i casi tipici ed è altamente affidabile
- **Flutter:** Flutter usa Dat, che mantiene Futter in testa nella selezione. Inoltre non esiste un bridge Javascript in FLutter per avviare le interazioni con i componenti nativi del dispositivo il che accelera drasticamente il tempo di esecuzione e la

velocità di sviluppo. Lo standard di animazione di FLutter è stato impostato a 60 fps il che indica le sue alte prestazioni. Infine poichè FLutter è compilato nel codice ARM sia per IOS che per Android per questo non deve mai affrontare problemi di prestazione.

- **Angular:** AngularJS è riconosciuto per le sue prestazioni moderate mentre si occupa di applicazioni complesse e dinamiche. Le app React e Flutter funzionano più velocemente delle app Angular delle stesse dimensioni. Tuttavia, alcune nuove versioni di Angular sono un po 'più veloci rispetto a React.

Interfaccia Utente

- **React Native:** React App Development utilizza librerie di terze parti poiché non dispone di una propria libreria di componenti dell'interfaccia utente. React Native Material Design e Shoutem sono due esempi di librerie dell'interfaccia utente accessibili agli utenti. React Native è simile all'utilizzo di HTML senza un framework CSS. Rispetto a Flutter, l'interfaccia utente di React Native si basa maggiormente sui componenti nativi sia per iOS che per Android. Fornisce inoltre un'esperienza utente (UX) più piacevole quando un utente accede al sistema operativo.

- **Flutter:** incorpora il Material Design e anche Cupertino design. Flutter ha i suoi singoli componenti dell'interfaccia utente, set di widget adattabili e design dei materiali insieme a un motore che aiuta a renderizzarli su IOS e Android.

- **Angular**: Angular ha uno stack tecnologico di materiali integrato. Viene fornito con molti componenti di progettazione dei materiali pre-costruiti. Ciò rende la configurazione dell'interfaccia utente estremamente agile e semplice.

Supporto della community

- **React Native:** React è stato rilasciato come open source su GitHub nel 2015 ed è il framework più popolare su Stack Overflow. È supportato da una comunità enorme, con oltre 89k stelle su GitHub, 58.4k utenti sul suo subreddit e una grande quantità di supporto su Stack Overflow. Questo è il motivo per cui hanno più librerie / plugin di terze parti rispetto a Flutter.

- **Flutter**: Il supporto della comunità Flutter può essere visto nelle sue stelle 98k su GitHub, subreddit utente 47.6k e Stack Overflow. Anche se Dart non ha ricevuto molta ammirazione nel sondaggio sugli sviluppatori di Stack Overflow, i primi post sul blog hanno dato un feedback positivo sull'uso di Flutter. Inoltre, la loro documentazione è molto approfondita e affronta tutte le domande pubblicate entro un lasso di tempo accettabile.

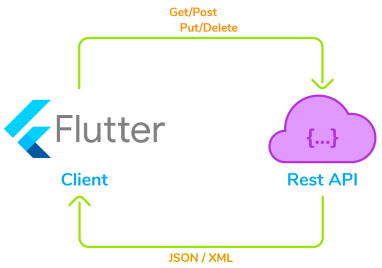
- **Angular:** Il supporto della comunità Flutter può essere visto nelle sue stelle 98k su GitHub, subreddit utente 47.6k e Stack Overflow. Anche se Dart non ha ricevuto molta ammirazione nel sondaggio sugli sviluppatori di Stack Overflow, i primi post

sul blog hanno dato un feedback positivo sull'uso di Flutter. Inoltre, la loro documentazione è molto approfondita e affronta tutte le domande pubblicate entro un lasso di tempo accettabile.

Al momento non esiste una risposta per definire quale sia migliore tra React, Flutter e Angular ma dipende solamente dalle esigenze aziendali e i casi d'uso. Nel nostra caso specifico tutti e 3 potevano soddisfare il requisito non funzionale di scalabilità ma uno solo poteva offrire prestazioni più alte per quanto riguarda i device nativi inoltre vi era un parametro aggiuntivo che per noi è risultato fondamentale per la scelta finale che era la conoscenza delle tecnologie, cioè tutti e 3 avevamo conoscenza ed esperienza in Flutter poichè ognuno di noi aveva seguito corsi, ottenuto certificati e implementando progetti a livello business quindi non si necessitava dell'apprendimento di una nuova tecnologie ma abbiamo cercato di soddisfare il time to market.

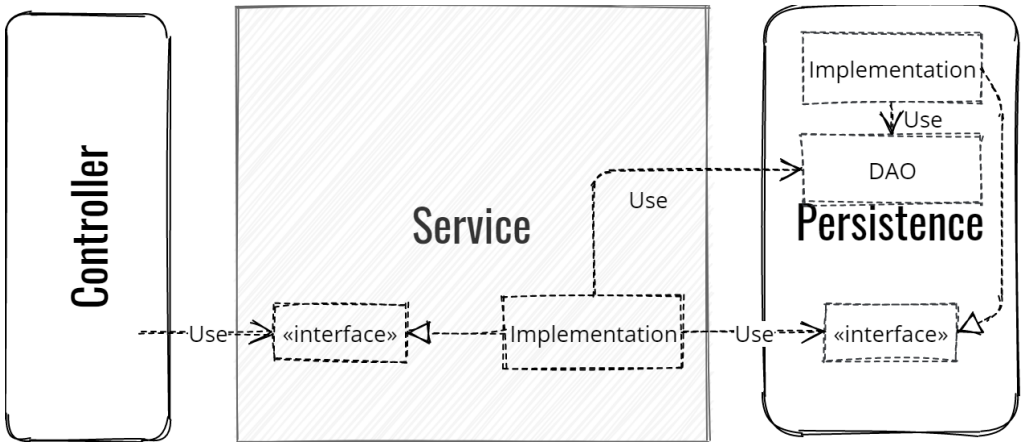
6.Nuova Struttura

Per quanto riguarda l'architettura del nuovo sistema, essa è rimasta client-server. Le differenze sostanziali sono le tecnologie e il design. La scelta finale delle tecnologie sono le seguenti: REST API per il lato server e Flutter per il lato client per le motivazioni che sono state citate precedentemente.



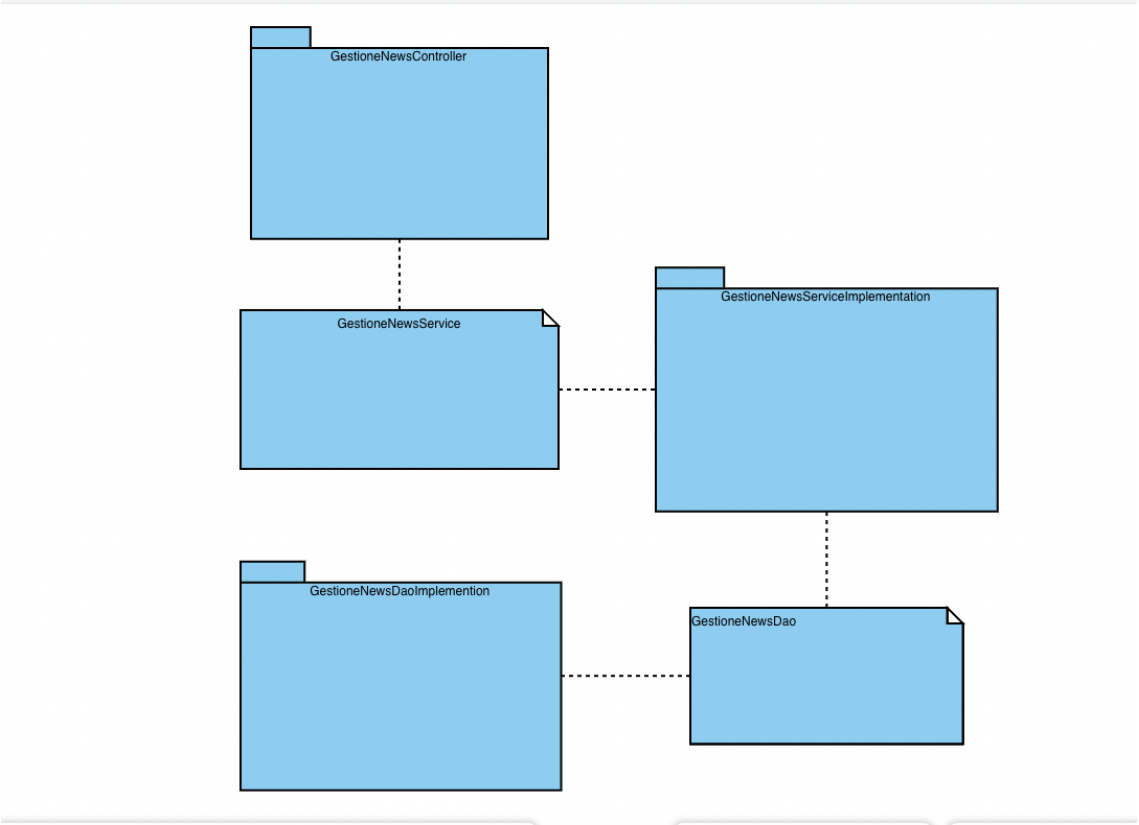
Nella figura 1.4 è riportato la struttura client-server attuale e le tecnologie utilizzate

Per quanto riguarda l'architettura utilizzeremo l'approccio MVC con un stile strutturale esagonale, o anche detta architettura delle porte e degli adattatori. E' un modello architetturale utilizzato nella progettazione del software, mira a creare sistemi basati su componenti applicativi che sono liberamente accoppiati e possono essere facilmente collegati al loro ambiente software tramite porte e adattatori. Questi componenti sono modulari e intercambiabili, il che migliora la coerenza dell'elaborazione e facilita l'automazione dei test. L'esterno dell'esagono (l'infrastruttura) è diviso in due parti virtuali, il lato sinistro e il lato destro. Sulla sinistra, si ha tutto ciò che interrogherà il dominio (il controller, il livello REST, ecc.); a destra tutto ciò che fornirà alcune informazioni/servizi al dominio (livello di persistenza, servizi di terze parti, ecc.).



Nella figura 1.5 è riportato la struttura esagonale lato server

La nostra servlet si trasforma in un controller che richiama il service che a sua volta va a richiamare il dao. La parte business logic è presente nel Service che può restituire l'oggetto oppure può invocare una throw per qualche errore. Adesso andremo a vedere il flow del codice e la comunicazione tra i vari componenti.



Nella figura 1.6 è riportato la struttura del nuovo sistema di quella che era una Servlet

E' possibile visionare il Class Diagram aggiornato che mostra il cambiamento spiegato nella figura precedente sommariamente su tutte le classi, sulla repository di GitHub al link: https://github.com/eliotest98/HardShipV2/tree/main/HardShipV2%20Documents.
Per quanto riguarda la fase di Alterazione della Migrazione sono state effettuate le seguenti attività:

- RE-CODE: è stata effettuato un Rephasing per il cambio di linguaggio di programmazione da JSP a Dart.
- RE-DESIGN: è stato effettuato il cambiamento di alcune parti delle Servlet con la trasformazione in REST API.

7. Rischi del nuovo sistema

Verranno definite precisamente le date di rilascio e definita la sequenza temporale del rilascio incrementale del sistema in quanto l'approccio incrementale richiede di definire delle tempistiche ben precise per schedulare tutte le attività.
I parametri per la rappresentazione del successo del progetto sono i seguenti:

- Rispetto delle date di rilascio incrementale del nuovo sistema.
  - Il tutto verrà gestito con branch con i vari rilasci che verranno mergiati nel momento in cui dovranno essere presenti sui nuovi dispositivi mobile.
- Rispetto dei requisiti da implementare.

8. Requisiti del sistema

Come già descritto precedentemente oltre ai requisiti funzionali derivati dal vecchio sistema siamo andati ad aggiungere quello che è un nuovo requisito non funzionale ovvero la scalabilità. Inoltre sono state definite le funzionalità minime, e si è passati alla **creazione del Prodotto Minimo Funzionante**. L'idea è quella di avere 3 fasi di sviluppo:

- PER LA VERSIONE 1.0
  - Il sistema permette all'utente di visualizzare gli album esistenti nel sistema.
  - Il sistema permette all'utente di verificare la disponibilità delle tipologie degli album proposti dal sistema.
  - Il sistema permette all'utente di riprodurre in anteprima i brani di un album.
  - Il sistema permette all'utente di visualizzare le news su artisti ed album presenti nel sito.
- PER LA VERSIONE 2.0
  - L'utente registrato può effettuare il login, inserendo le proprie credenziali per accedere al sistema.
  - L'utente può registrarsi.
- PER LA VERSIONE 3.0
  - Il sistema permette all'utente che ha effettuato il login, di selezionare un album ed aggiungerlo al carrello per effettuare l'acquisto.
  - Il sistema permette all'utente che ha effettuato il login, di accedere all'area carrello per visualizzare gli album aggiunti in precedenza.
  - Rimuovere album dal carrello.
  - Il sistema permette all'utente che ha effettuato il login, di eliminare gli album presenti nella lista del carrello.
  - Il sistema permette all'utente che ha effettuato il login di recensire un album lasciando un commento.

- Il sistema permette all'utente di visualizzare i feedback relativi agli album, lasciati da altri utenti precedentemente.
- L'utente può richiedere l'inserimento, all'interno del sito, di un album a cui è interessato per l'acquisto, ma non ancora disponibile.
- L'utente termina la sessione disconnettendosi dal sistema.

9. Riutilizzo delle componenti

Una volta definito l'approccio e la struttura del sistema, abbiamo cercato di capire quali fossero le componenti e le porzioni di codice che potessero essere riutilizzate nel nostro progetto.

Le componenti che sostanzialmente sono state riutilizzate sono:

- Le **entity** per il database: poiché il database non ha subito nessuna variazione dal vecchio al nuovo sistema, nel primo step sono state importate le entity che ci hanno permesso di velocizzare la fase iniziale senza dover riscrivere le diverse entità presenti in quanto uguali.
- Le **query** innestate all'interno delle Servlet: ciò ci ha permesso di velocizzare il processo di implementazione dei DAO sempre nel primo step del progetto. Il processo di estrazione dalle servlet consisteva nel considerare quale fosse la servlet a svolgere una determinata azione in modo da essere sicuri del comportamento della query testando prima la funzionalità sul vecchio sistema per poi importarlo sul nuovo sistema. In questo modo anche in caso di eventuali difetti nel codice o nella query sarebbe stato possibile individuare immediatamente il bug.
- **Casi di test**: alcuni casi di test effettuati nel vecchio sistema a cui sono stati aggiunti altri casi di test per poter effettuare un test delle funzionalità del sistema. I test sono stati selezionati basandosi anche sulle query che dovevano essere riutilizzate in modo da poter avere 2 feedback uno con i test d'unità e un altro con un test manuale del vecchio sistema.

10. Testing

Il testing è stato eseguito seguendo quello che è un approccio incrementale, facendo in modo che i test venissero sviluppati in contemporanea all'implementazione delle versioni del sistema.

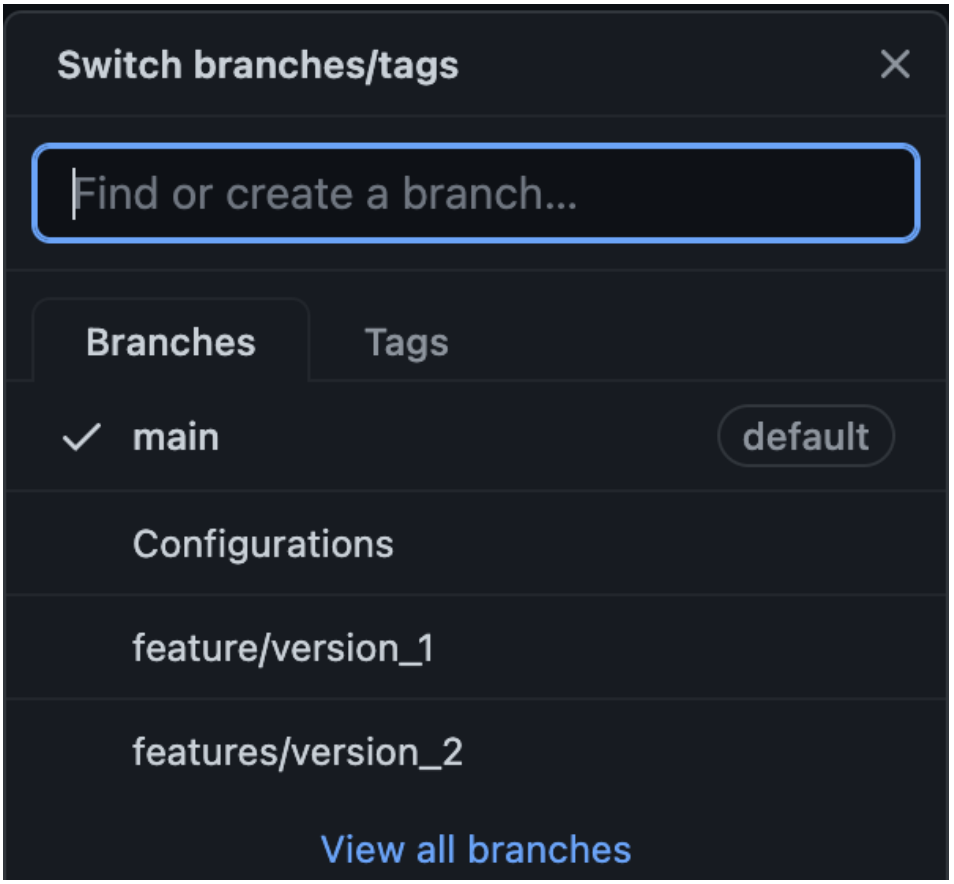
E' stato effettuato il testing in tre periodi diversi:

- Nel primo periodo è stato testato il vecchio sistema per migliorare la conoscenza di esso e testare le varie funzionalità già presenti utilizzando quelli che sono i test d'unità, ciò ci ha permesso di testare la maggior parte del sistema.
- Nel secondo periodo è stato testato il sistema al rilascio della versione 1.0 utilizzando sempre test di unità per le singole componenti, sono stati testati i Servizi e i Controller utilizzando la stessa tecnologia del vecchio sistema, ed inoltre sono stati eseguiti i test del vecchio sistema in aggiunta ad i nuovi.
- Nel terzo periodo è stato testato il sistema al rilascio della versione 2.0 utilizzando anche in questo caso test d'unità per le singole componenti, inoltre sono stati ripetuti i vecchi test del primo e del secondo periodo.

Ad ogni periodo è stato eseguito quello che è il coverage del sistema per stimare il numero di funzionalità testate per evidenziare eventuali errori.

11. Definizione Strategia

La strategia che più si avvicina alle nostre esigenze è l'approccio incrementale anche detto Chicken Little, in quanto ci permette di creare un nuovo sistema in esecuzione parallelo al vecchio rispetto a creare il nuovo sistema e sostituirlo tutto in una volta. Molto rischioso. In primo luogo siamo andati dapprima ad implementare la versione V1 staccando un branch. Successivamente dopo la prima versione è stato effettuato il merge sul main da dove poi verrà rilasciata la prima versione del nostro sistema ovvero l'applicazione FLUTTER e il SERVER REST con un minimo di prodotto funzionante(MVP), nel frattempo siamo andati avanti con la nuova implementazione creando un nuovo branch, dove sono stati poi implementati i requisiti necessari per la V2 ed eseguito lo stesso procedimento per il rilascio della seconda versione. Poi proseguendo per la V3 ovvero l'ultima fase sarà anche in questo caso staccato un branch e implementate le varie feature per poi effettuare il merge sul main ottenendo così il software allo stadio finale pronto per il rilascio.



Nella figura 1.7 è riportata la repository suddivisa per la fase iniziale e le prime 2 fasi di sviluppo

12. Fattibilità della Strategia

La strategia quindi ci permetterà di suddividere il vecchio sistema in tante funzionalità e incrementalmente sviluppare una singola funzionalità alla volta con una scadenza prefissata. A mano a mano che sarà implementata una funzionalità si sostituirà la funzionalità del vecchio sistema fino al totale spegnimento.

13. Determinare le risorse necessarie

Per poter realizzare la strategia servirà una fase di definizione e di suddivisione delle varie funzionalità del vecchio sistema in modo da poter definire le tempistiche di realizzazione delle singole funzionalità e del termine della migrazione.