Documentazione di progetto

Corso di informatica III
Modulo Progettazione e algoritmi

Studenti
MAT 1053000 - Luca Rillosi
MAT 1053031 - Emanuele Rota

Docente
Prof.ssa Patrizia Scandurra

Iterazione 0

Introduzione - Panoramica

L’obiettivo primario è quello di fornire a chi acquista un’esperienza che si avvicina il più possibile a quella dell’acquisto online di un libro nuovo, fornendo al tempo stesso una vetrina in cui gli studenti possono mettere in vendita i propri testi usati.

Analisi dei requisiti

Analisi del contesto

Durante la carriera universitaria si è rivelato essenziale avere accesso a numerosi libri e testi accademici per supportare gli studi. Sin da subito però, è apparsa evidente la mancanza di un hub di riferimento per la ricerca, il prestito e la compravendita di libri universitari.

Pur esistendo numerosi servizi online per la consultazione del catalogo della biblioteca, l’acquisto di libri nuovi, usati, e la pubblicazione di annunci di vendita, tali servizi rimangono separati costringendo lo studente interessato alla consultazione di un testo a visitare e monitorare con cadenza anche giornaliera un numero sempre crescente di siti.

L’obiettivo che si pone questo progetto è di creare un hub di riferimento per lo studente, dove tramite un’unica ricerca lo studente ottiene risultati aggregati provenienti da tutti i servizi a lui disponibili ed aggiornati in tempo reale.

Studio di fattibilità

Per realizzare le funzionalità della web app sono necessari i seguenti elementi:

- Interfacciamento con siti di e-commerce: la app deve essere in grado di ottenere in tempo reale dati riguardo i prezzi e la disponibilità degli articoli richiesti nei vari store online. Questo interfacciamento può essere ottenuto tramite API (Amazon product API) oppure via scraping del sito web (Web scraper)
- Interfacciamento con il sistema bibliotecario: Questo è necessario per presentare all’utente informazioni riguardo la disponibilità del libro richiesto in una biblioteca nelle vicinanze. Può essere ottenuto mediante scraping oppure tramite interfacciamento con le stesse API utilizzate dal sito ufficiale del sistema bibliotecario.
- Piattaforma di multivendor e-commerce: è necessario creare un’infrastruttura che permetta all’utente di registrarsi sia come venditore sia come acquirente. A tal fine è possibile utilizzare una soluzione già pronta e solamente da configurare (CMS) oppure è possibile svilupparne una custom, che si integri perfettamente con gli altri servizi forniti dalla app e consenta di ottenere il massimo livello di customizzazine possibile.

Casi d’uso

Descrizione generale

La app fornisce funzioni differenti a seconda del tipo di utente che lo visita (amministratore, utente registrato o ospite).

Un utente amministratore può aggiungere nuovi libri (UC1) specificando titolo, codice ISBN, casa editrice, autori ed altri metadati relativi al libro da aggiungere. Questi campi sono precompilati grazie all’aiuto di uno scraper che ottiene metadati da vari database di libri online.

Qualsiasi utente (registrato o ospite) può visualizzare l’elenco ed effettuare una ricerca libera per titolo, ISBN o casa editrice (UC2).

Una volta trovato il libro desiderato si viene portati alla pagina relativa al libro. Qui sono mostrate le informazioni aggregate dalle varie sorgenti dati.

L’utente può inserire il proprio annuncio (UC3): durante la procedura, se l’utente non è loggato verrà rimandato alla pagina di login, mentre se lo è al completamento della procedura l’annuncio verrà aggiunto alla pagina del libro.

Se l’utente ha effettuato il login (UC6), può visitare la pagina “profilo” in cui può visualizzare le proprie informazioni (UC4) All’interno della stessa pagina è presente un elenco con tutti gli annunci pubblicati dall’utente, con un pulsante “Elimina” che permette di rimuovere ciascun annuncio singolarmente (UC5).

Inserimento nuovo libro nel DB (UC1)

Descrizione

Un utente amministratore può aggiungere nuovi libri al database. Durante l’immissione dei dati è aiutato dall'auto completamento automatico che fornisce metadati ottenuto da sorgenti esterne.

Requisiti coperti

A7 (+ B1)

Attori

Utente amministratore

Precondizioni

L’utente è loggato come amministratore

Passi principali

(a) L’utente inserisce un termine di ricerca nella barra di autocompletamento e clicca “cerca”

(b) L’auto completamento riempie i dati di una maschera “nuovo libro” con i metadati trovati

(c) L’utente verifica e corregge i dati, poi preme “invia”

Situazioni eccezionali

Dati inseriti non validi

Postcondizioni

Il libro è aggiunto al database

Ricerca libro (UC2)

Descrizione

Qualsiasi utente (registrato o ospite) può effettuare una ricerca libera per titolo, ISBN o casa editrice e visualizzare i dati relativi al libro desiderato.

Requisiti coperti

A3

Attori

Utente registrato o ospite

Precondizioni

/

Passi principali

(a) L’utente inserisce la parola chiave nella barra di ricerca e clicca su un libro nella tendina “risultati”

(b) L’utente viene portato alla pagina di aggregazione annunci per il libro richiesto

Situazioni eccezionali

Nessun libro trovato

Postcondizioni

/

Inserimento annuncio (UC3)

Descrizione

L’utente registrato può pubblicare un annuncio di vendita specificando libro, condizioni, prezzo e dati dove poter essere contattato dall’acquirente

Requisiti coperti

A4

Attori

Utente registrato o ospite

Precondizioni

L'utente ha trovato il libro che desidera vendere (UC2) e sta visualizzando la relativa pagina libro.

Passi principali

(a) L'utente clicca su "Vendi il tuo usato"

(b) L'utente inserisce le condizioni del libro, il prezzo ed eventuali note aggiuntive

(c) L'utente clicca "Avanti"

(d) L'utente deve inserire un'e-mail dove poter essere contattato e da cui potrà gestire i suoi annunci

(e) L'annuncio viene inserito e l'utente viene informato dell'esito dell'operazione

Percorsi alternativi

(d1a) se l'utente è già loggato si salta direttamente allo step (e) e come e-mail viene utilizzata quella inserita in precedenza dall'utente

(d1b) Se l'utente non è loggato ma inserisce una email già registrata nel DB, viene portato alla schermata login (UC6). Dopo il login si prosegue allo step (d)

Situazioni eccezionali

/

Postcondizioni

L'annuncio è inserito nel database, l'utente è registrato nel sistema

Login (UC6)

Descrizione

L'utente può autenticarsi nel sistema utilizzando le credenziali inserite in fase di registrazione

Requisiti coperti

A2

Attori

Utente

Precondizioni

Utente non loggato, utente registrato

Passi principali

(a) L’utente inserisce le credenziali nel sistema

(b) L’utente preme sul pulsante login

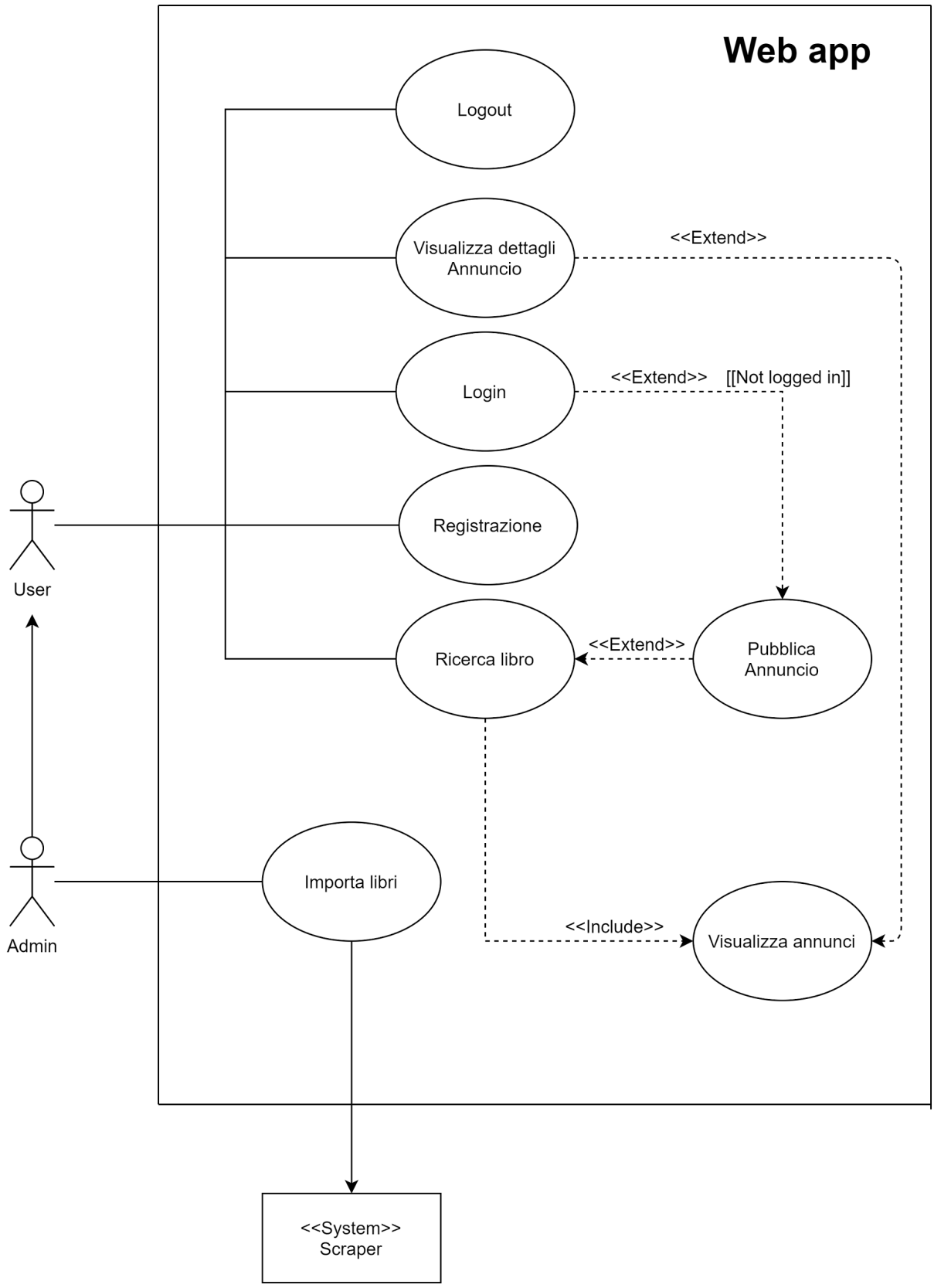
(c) Il sistema risponde con un messaggio in base all’esito dell’autenticazione

Situazioni eccezionali

Credenziali non valide

Postcondizioni

Utente autenticato



Casi d’uso iterazione 0

Specifiche

Specifiche funzionali

Sono state definite specifiche divise in priorità alta, media e bassa. Le specifiche con alta priorità consistono in quelle funzioni chiave necessarie al corretto funzionamento della prima versione dell'app.
Le Specifiche con priorità media o bassa consistono nelle funzionalità aggiuntive non strettamente necessarie che saranno implementate nelle successive versioni rilasciate.

Alta priorità

Cod	Titolo
A1	Registrazione utente
A2	Login utente
A3	Ricerca libro
A4	Pubblicazione annuncio
A5	Rimozione annuncio
A6	Logout utente
A7	Inserimento libro

Media priorità

Cod	Titolo
M1	Aggiornamento dati utente
M2	Visualizzazione articoli e-commerce
M3	Disponibilità in biblioteca
M4	Notifiche nuovo annuncio su Telegram
M5	Deduplicazione libri

Bassa priorità

Cod	Titolo

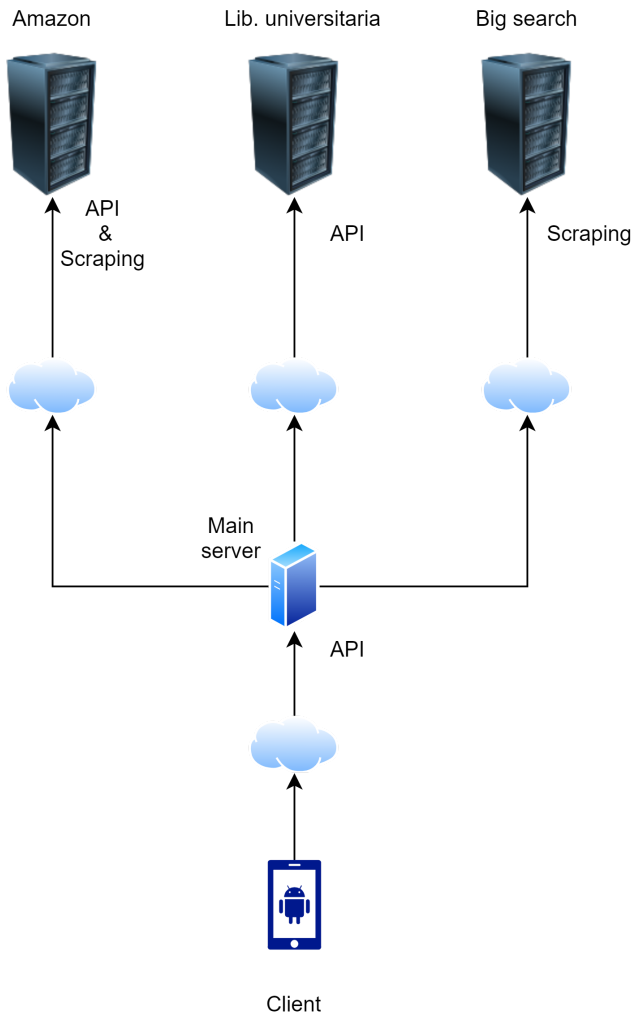
B1	Autocompletamento metadati nuovo libro
B2	Raggruppamento articoli simili
B3	Rimozione libri duplicati

Architettura

Diagramma architetturale

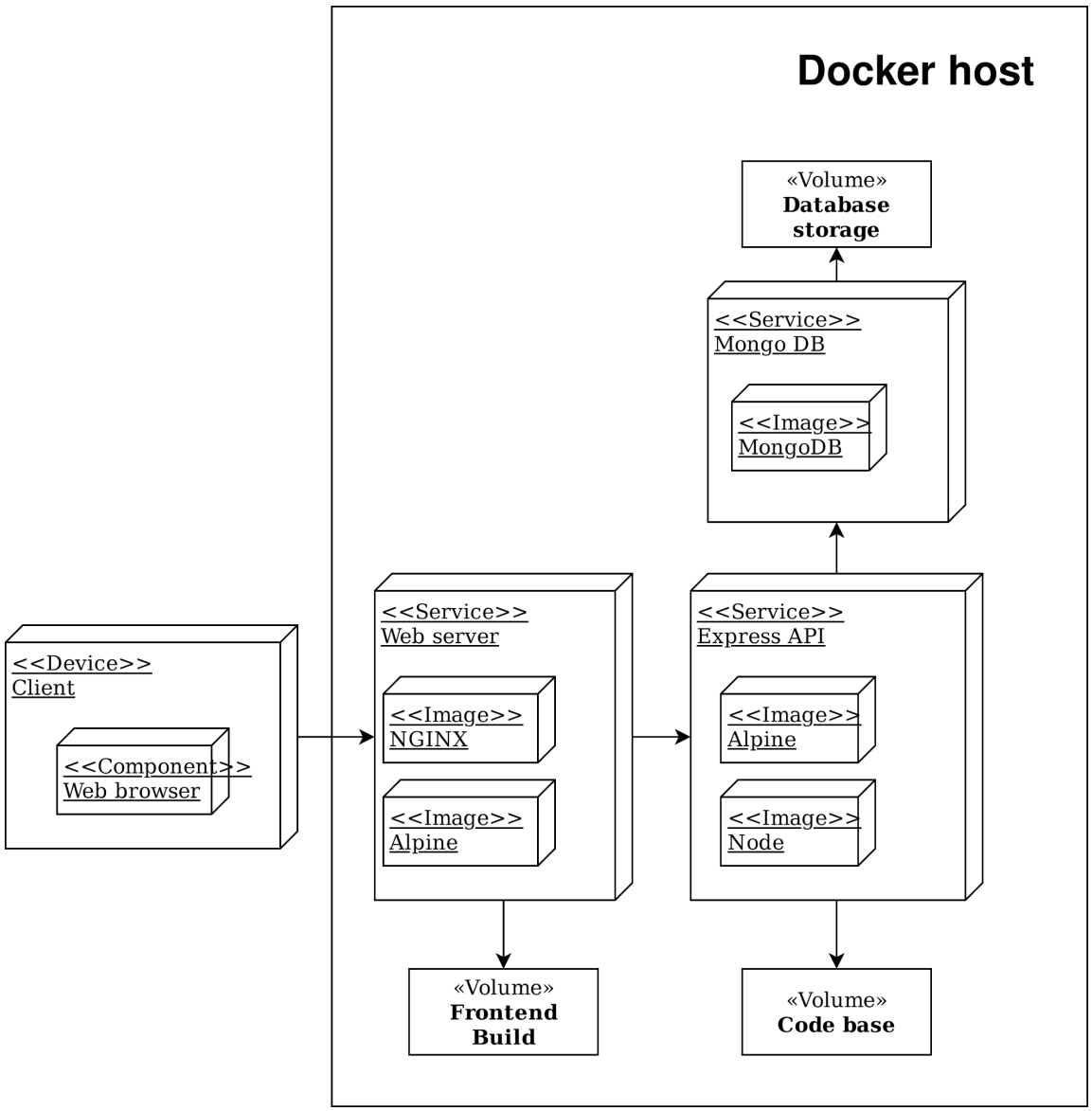
L'architettura utilizzata è di tipo client-server a due livelli:

- Il primo livello è costituito dal client (ReactJS web-app) che comunica con il server tramite Restful API (protocollo HTTPS)
- Il secondo livello è rappresentato dal server, che si interfaccia con Amazon, Libreria universitaria e il sito della biblioteca al fine di aggiornare il catalogo. L'interfacciamento avviene tramite Restful API (HTTPS) o mediante scraping di pagine web (HTTPS).



Deployment Diagram

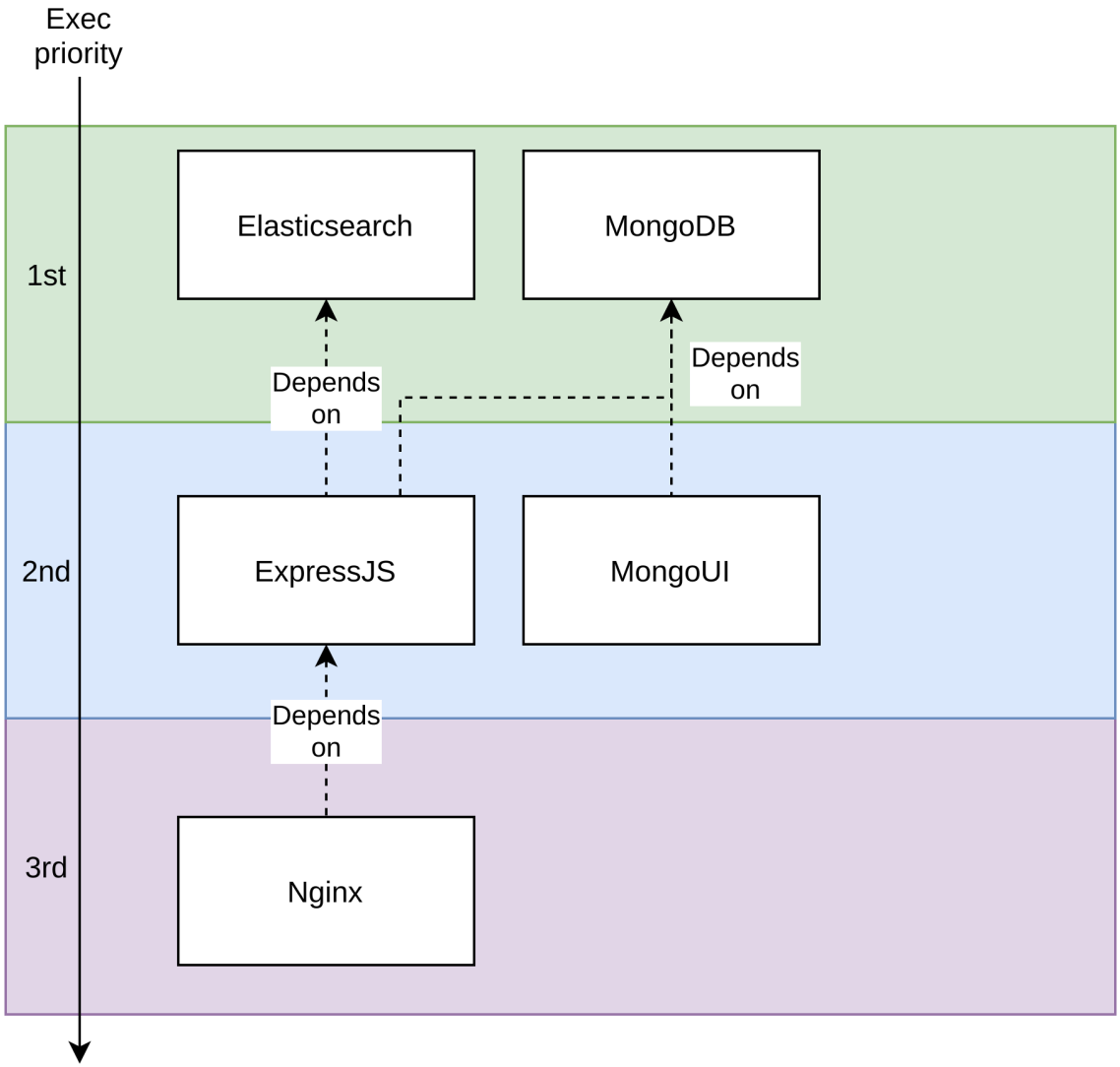
Il cuore del sistema è rappresentato dal server centrale, indicato con “Docker host”.
Tale server è organizzato seguendo il paradigma dell’architettura a microservizi, che consente di impacchettare ogni componente fondamentale della app in un contenitore (container) che previene i conflitti di dipendenze e facilita l’interazione tra le varie componenti del sistema.
Il device client si collega mediante un web browser all’indirizzo del sito, e in questo modo si interfaccia con il web server presente nel container omonimo sul server.
Il web server è costituito da Nginx, un server HTTP che in questo caso è utilizzato come reverse proxy per consentire l’interfacciamento tra il client e l’API nel container “Express API”.
Il container Express API si interfaccia inoltre con il container MongoDB (che costituisce la memoria persistente in cui l'app memorizza tutti i dati).



Deployment diagram iterazione 0

Ciclo di vita dei container (micro servizi)

Nel diagramma sottostante viene mostrato il ciclo di vita dei microservizi implementati, in particolar modo è possibile vedere l'ordine di lancio di ciascun container, e le sue relative dipendenze nei confronti degli altri.



Ciclo di vita dei container

Toolchain

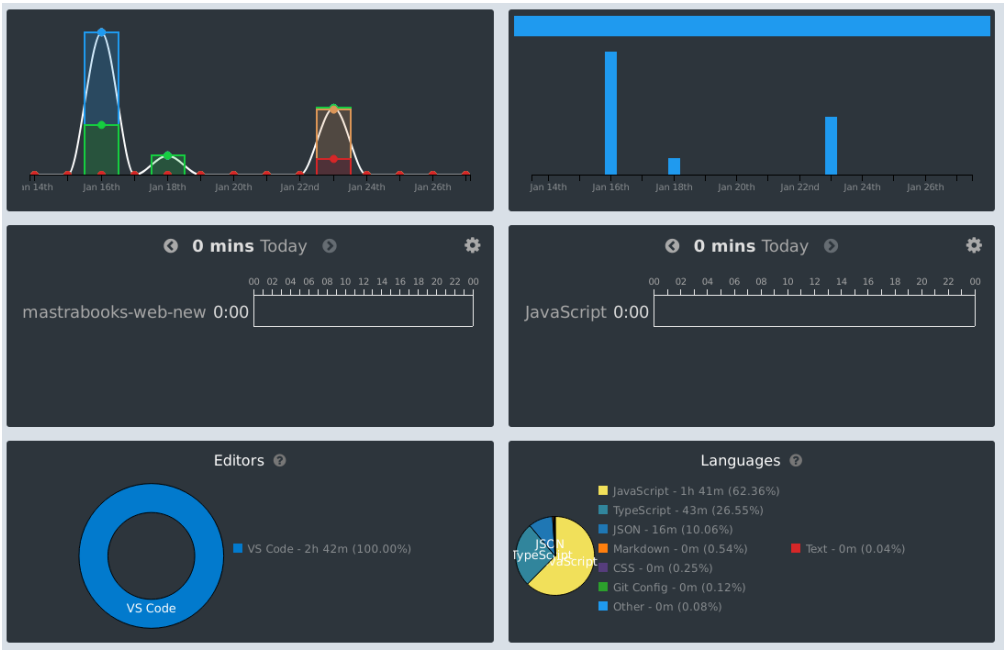
Descrizione generale della toolchain completa

Tecnologia / piattaforma	Descrizione
VScode	Editor utilizzato, ambiente di sviluppo
Draw.io	Software utilizzato per disegnare i diagrammi
GitLab	Piattaforma scelta per il versioning, basata su GIT
Wakatime	Tool integrato in Vscode per il monitoraggio delle ore di lavoro
Docker & Docker-compose	Tecnologia per la realizzazione di microservizi
Express js	Libreria basata su NodeJS per lo sviluppo di API
React	Libreria JavaScript per lo sviluppo frontend
ESlint	Tool utilizzato per l'analisi statica del codice
Prettifier	Tool utilizzato per la standardizzazione del codice (beautifier)
Npm-audit	Analisi statica delle vulnerabilità
Insomnia Designer	Programma utilizzato per l'esecuzione di test dinamici
Bash script	Script linux utilizzati per il deploy automatico del sistema
Google DOCs / Drive	Piattaforma di editing per la stesura della documentazione e condivisione di files
OVH hosting	Hosting VPS usata per la production del sito
CertBot/Let'sEncrypt	Auto rinnovo dei certificati SSL
MongoDB compass	GUI di amministrazione database MongoDB
Figma	Software grafico per la generazione dei mockup del sito

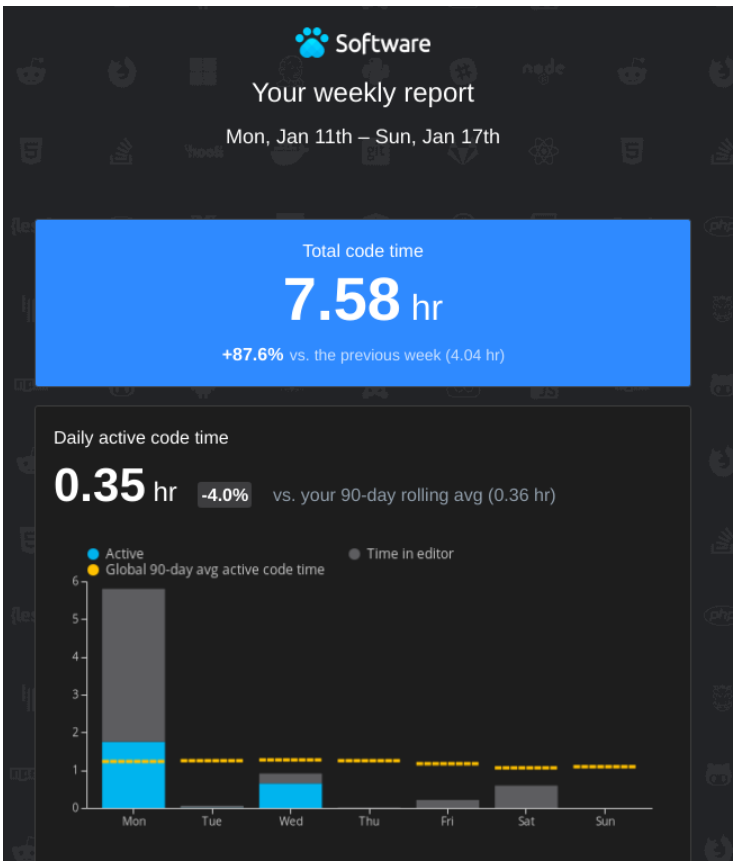
Dettaglio dei tool di maggior rilievo

Wakatime

Per il resoconto delle ore assegnate al progetto ci siamo affidati al tool wakatime, il quale genera dei report che mostrano chiaramente la gestione del tempo e permettono di tener traccia delle ore effettivamente attribuite alla fase di coding.



Report dell’attività diviso in linguaggi, IDE utilizzati ed altre statistiche



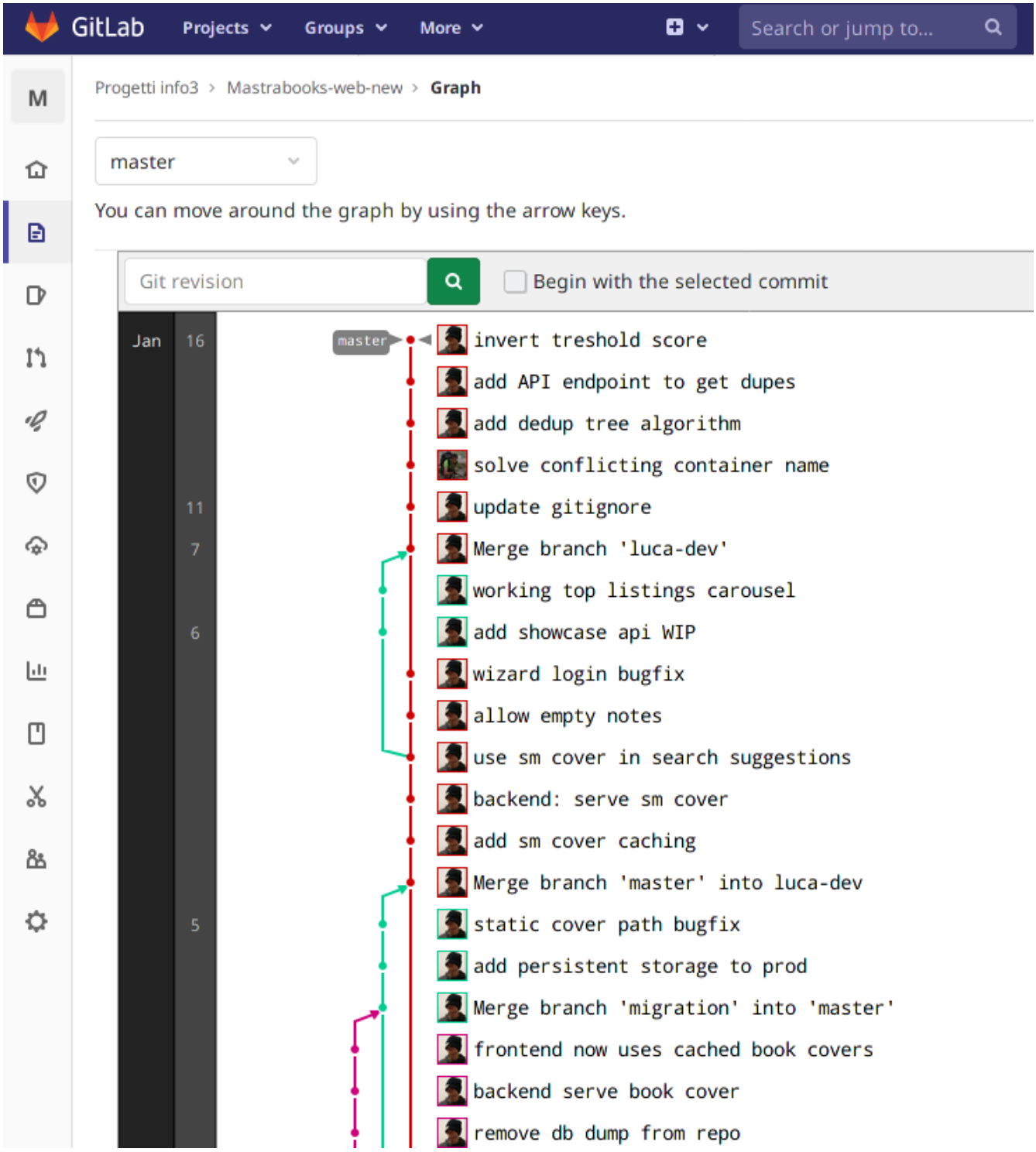
Report settimanale delle ore di lavoro

Report dettagliato delle performance lavorative

Visualizzazione della distribuzione delle ore nella schedule settimanale (vista calendario)

Gitlab

Piattaforma di versioning basata sul famoso software open source GIT.
Nelle seguente immagine vengono mostrati le ultime commit eseguite sul progetto, con un ulteriore vista che mostra l'andamento dei vari branch. Tale grafico risulta particolarmente utile per analizzare l'andamento dei vari branch attualmente in uso

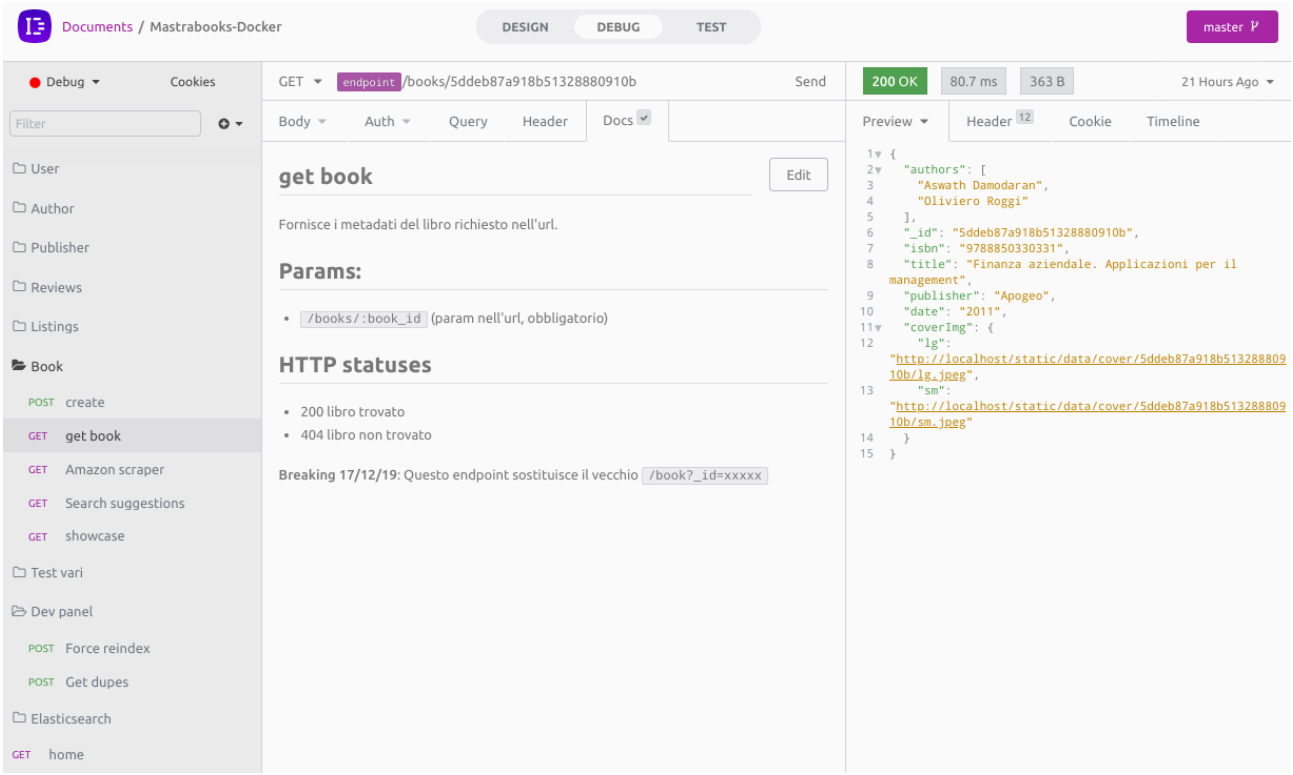


Commit sui branch della repository GitLab

Insomnia designer

Abbiamo utilizzato Insomnia designer per 2 scopi:
Per effettuare la stesura della documentazione di tutte le API esposte dal nostro sito

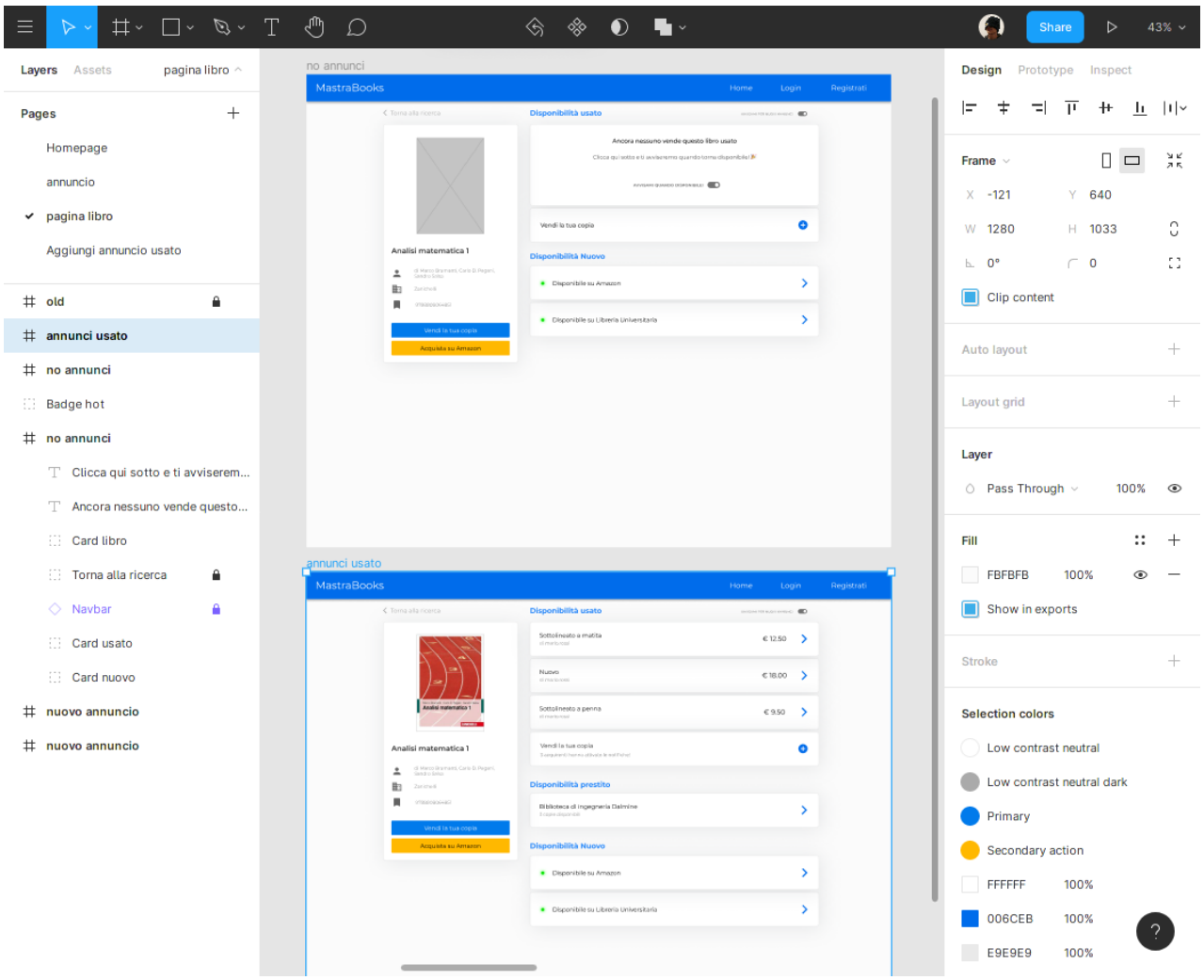
Per effettuare test dinamici con relativi report per verificare il corretto funzionamento dell'API dopo le ultime modifiche effettuate.



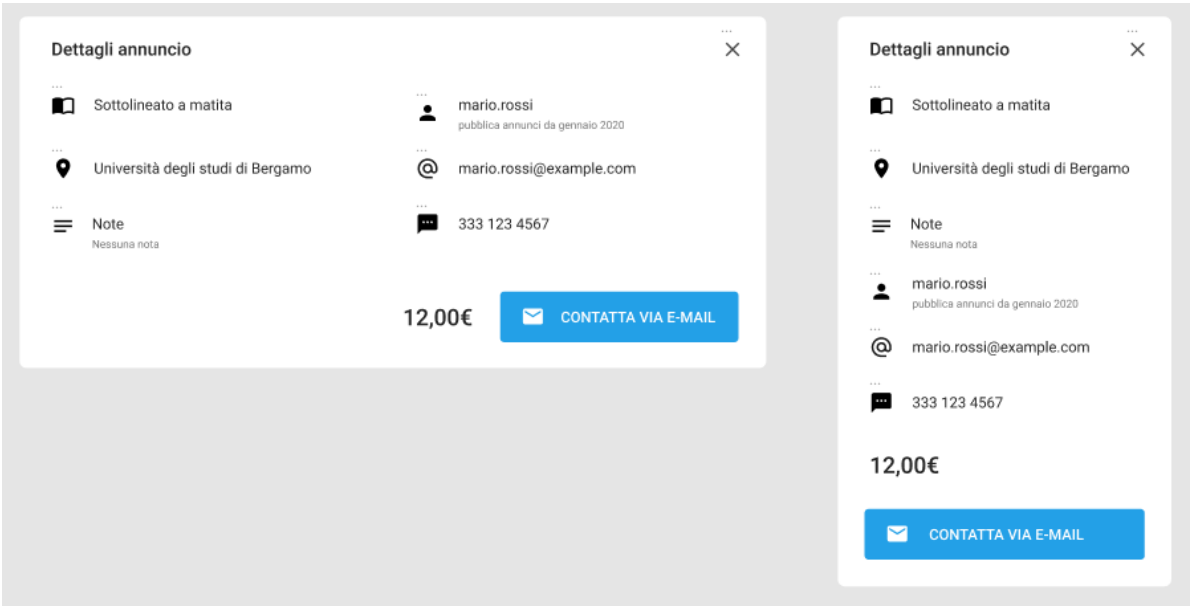
Documentazione di un API endpoint con insomnia designer

Figma mockup

Per garantire una maggiore scorrevolezza nelle varie fasi di sviluppo del progetto, abbiamo deciso di integrare nella nostra tool chain FIGMA.
Si tratta di un tool per la generazione di MOCKUP, i quali una volta realizzati permettono una più veloce stesura della parte front end del sistema, rendendo più facile e veloce il passaggio da prototipo a prodotto finito.



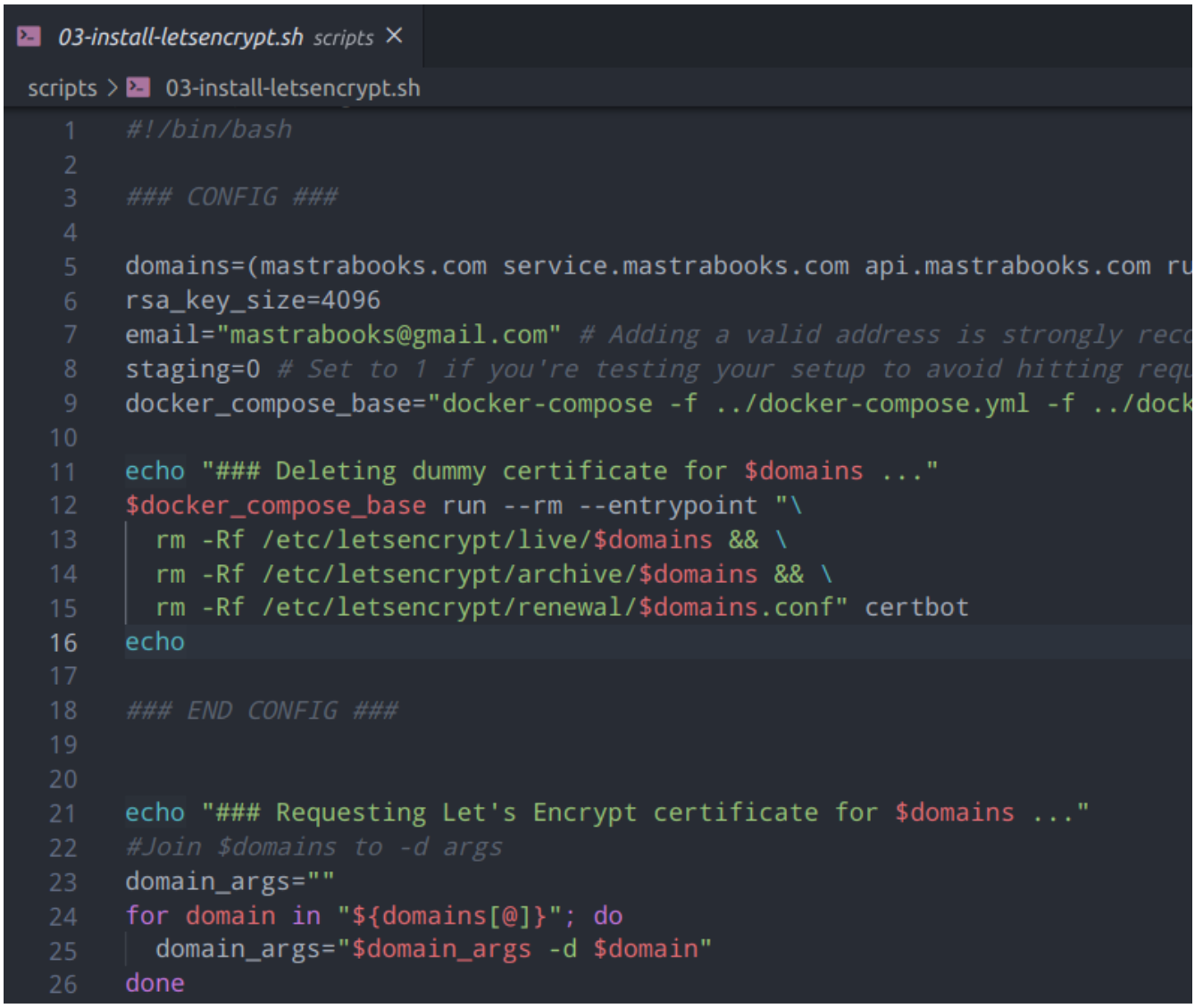
Due iterazioni dello sketch della pagina libro



Sketch della finestra dei dettagli di un annuncio (con vista desktop e mobile)

DevOps: Bash scripts

Per ottimizzare e velocizzare le operazioni di setup e deploy del servizio, abbiamo definito le operazioni da eseguire sotto forma di script in linguaggio bash. Questi script sono eseguibili tramite terminale ed assicurano che l'infrastruttura sia configurabile ed eseguibile in modo semplice e veloce.



Script che gestisce l'installazione del certificato SSL per HTTPS

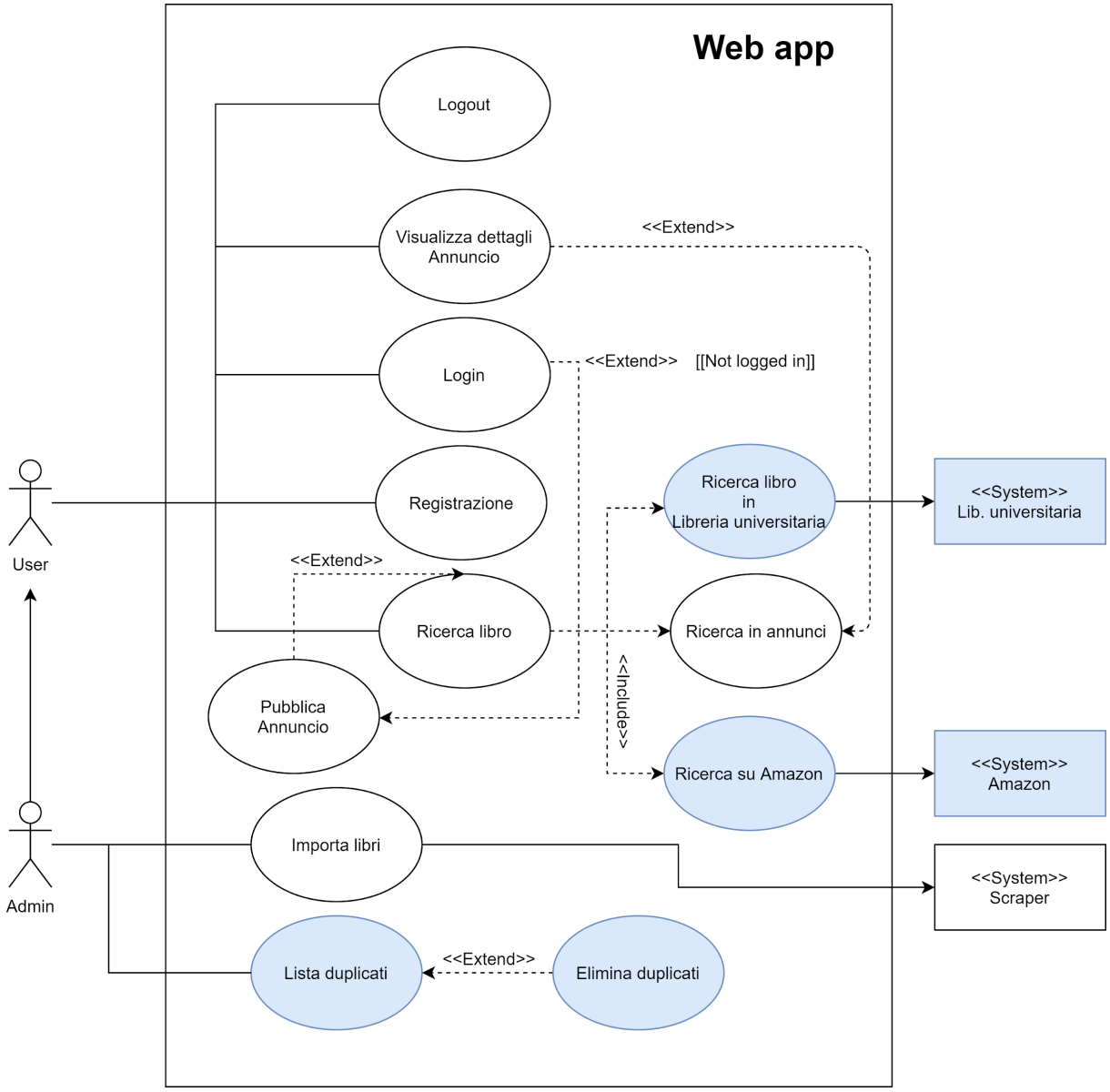
Iterazione 1

Nuovi casi d’uso

Per l’iterazione 1 abbiamo deciso di implementare i seguenti nuovi casi d’uso:

- Ricerca di un libro su libreria universitaria
- Ricerca di un libro su Amazon
- Creazione della lista libri duplicati

Dal diagramma sottostante è possibile analizzare l’evolversi dello stato del progetto rispetto alla iterazione precedente, in particolare si può notare i nuovi casi d'uso implementati di colore azzurrino, mentre la restante parte non evidenziata rispecchia l’architettura già in produzione prima della implementazione delle nuove features.



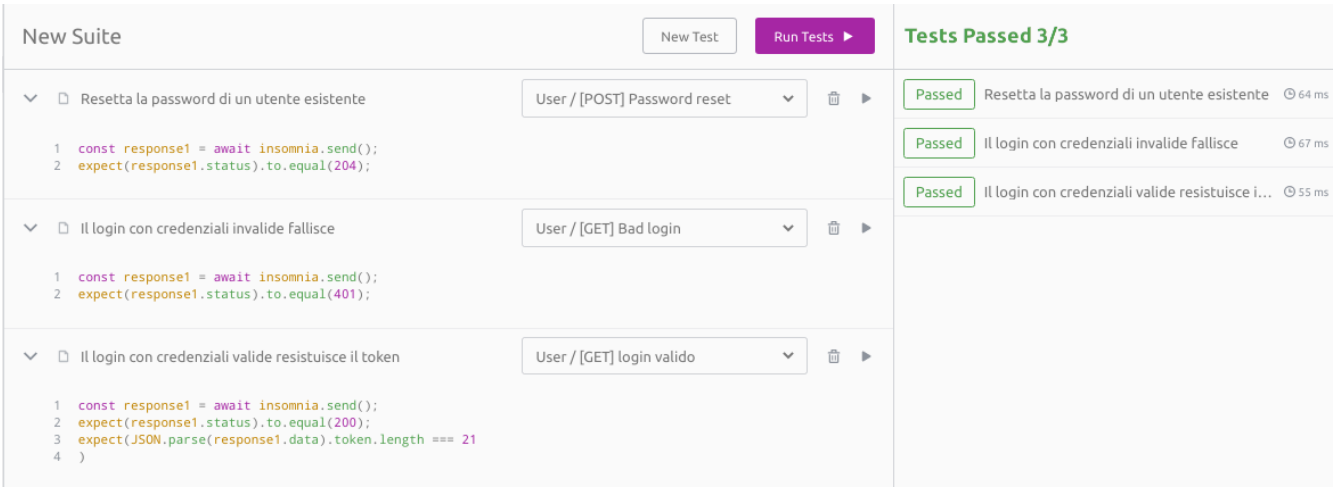
Casi d’uso iterazione 1

Test dinamico

Per l'iterazione 1 abbiamo implementato i seguenti test

- Resetta la password di un utente già esistente
- Login con credenziali errate
- Login con credenziali valide

Viene in seguito mostrato il report di insomnia designer una volta che è stato lanciato è eseguita con successo la serie di test definiti



Suite di test iterazione 1

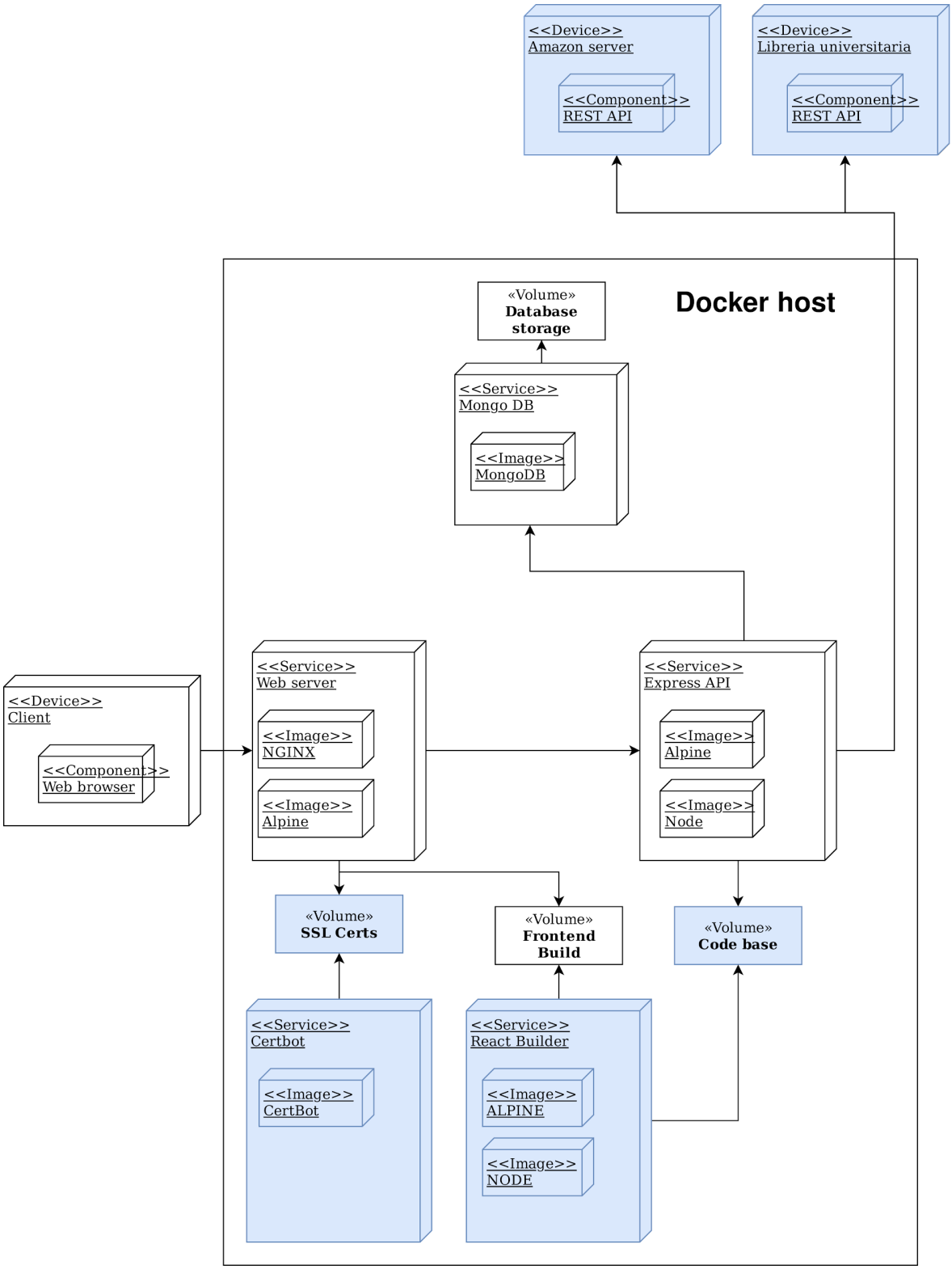
Nuova architettura

Deployment diagram

Nell'iterazione 1 si è inoltre attuata una naturale evoluzione dell'architettura che ha permesso di rendere il sito più sicuro ed al tempo stesso agevole da aggiornare. Sono stati introdotti i seguenti miglioramenti:

- Mediante l'aggiunta del microservizio Certbot, è stato possibile automatizzare la procedura di ottenimento di un certificato SSL per il protocollo HTTPS criptato. Tale container rimane in background fino a quando il certificato non necessita di un rinnovo. In tale occasione, effettua una ACME challenge con i server Let'sEncrypt ed installa il nuovo certificato senza causare downtime.
- E' stato aggiunto un microservizio (in apposito container "React builder") che, lanciato al momento del deploy in produzione, effettua la build del codice frontend (React) in maniera totalmente autonoma e trasparente, agevolando il task del deploy delle nuove versioni dell'app.

Di seguito è riportata la nuova architettura aggiornata, con i nuovi micro servizi implementati evidenziati in azzurro.



Deployment diagram iterazione 1

Analisi statica

Per il mantenimento di una buona qualità del codice ed il monitoraggio di un'eventuale presenza di vulnerabilità o bugs nelle librerie utilizzate (dependencies) abbiamo utilizzato i due plugin più diffusi nel mondo Javascript: prettier e npm-audit.
Npm audit scansiona tutte le librerie incluse nel codice e le confronta con un database di definizioni, generando un report come quello riportato in figura:

Low	**Prototype Pollution**
Package	ini
Dependency of	sharp
Path	sharp > prebuild-install > rc > ini
More info	https://npmjs.com/advisories/1589

Manual Review
Some vulnerabilities require your attention to resolve

Visit https://go.npm.me/audit-guide for additional guidance

Low	**Prototype Pollution**
Package	yargs-parser
Patched in	>=13.1.2 <14.0.0 || >=15.0.1 <16.0.0 || >=18.1.2
Dependency of	migrate-mongoose
Path	migrate-mongoose > yargs > yargs-parser
More info	https://npmjs.com/advisories/1500

found 9 **low** severity vulnerabilities in 759 scanned packages
run `npm audit fix` to fix 4 of them.
4 vulnerabilities require semver-major dependency updates.
1 vulnerability requires manual review. See the full report for details.

Esempio di report di npm-audit

Il plugin prettier si integra con l'IDE scelto (Nel nostro caso Visual Studio Code) e formatta automaticamente il codice scritto seguendo le specifiche ufficiali di linting definite dallo standard ECMA. Il plugin è configurabile mediante un file chiamato .prettierc ed inserito all'interno della root del progetto, tuttavia ci siamo attenuti alle guidelines della documentazione ufficiale che consigliano di utilizzare ove possibile le impostazioni di default

per garantire l’omogeneità della formattazione del codice JS con tutti gli altri progetti che ne fanno uso. segue uno screenshot del codice pre e post salvataggio.

```
// App
const app = express().use(express.json()).use(morgan(EXPRESS_LOG_FORMAT)).use(passport.initialize()).use(cors());
```

Codice di inizializzazione backend (illeggibile, riga lunga e confusionaria)

```
// App
const app = express()
  .use(express.json())
  .use(morgan(EXPRESS_LOG_FORMAT))
  .use(passport.initialize())
  .use(cors());
```

Codice dopo il salvataggio e beautifying, molto più chiaro e leggibile

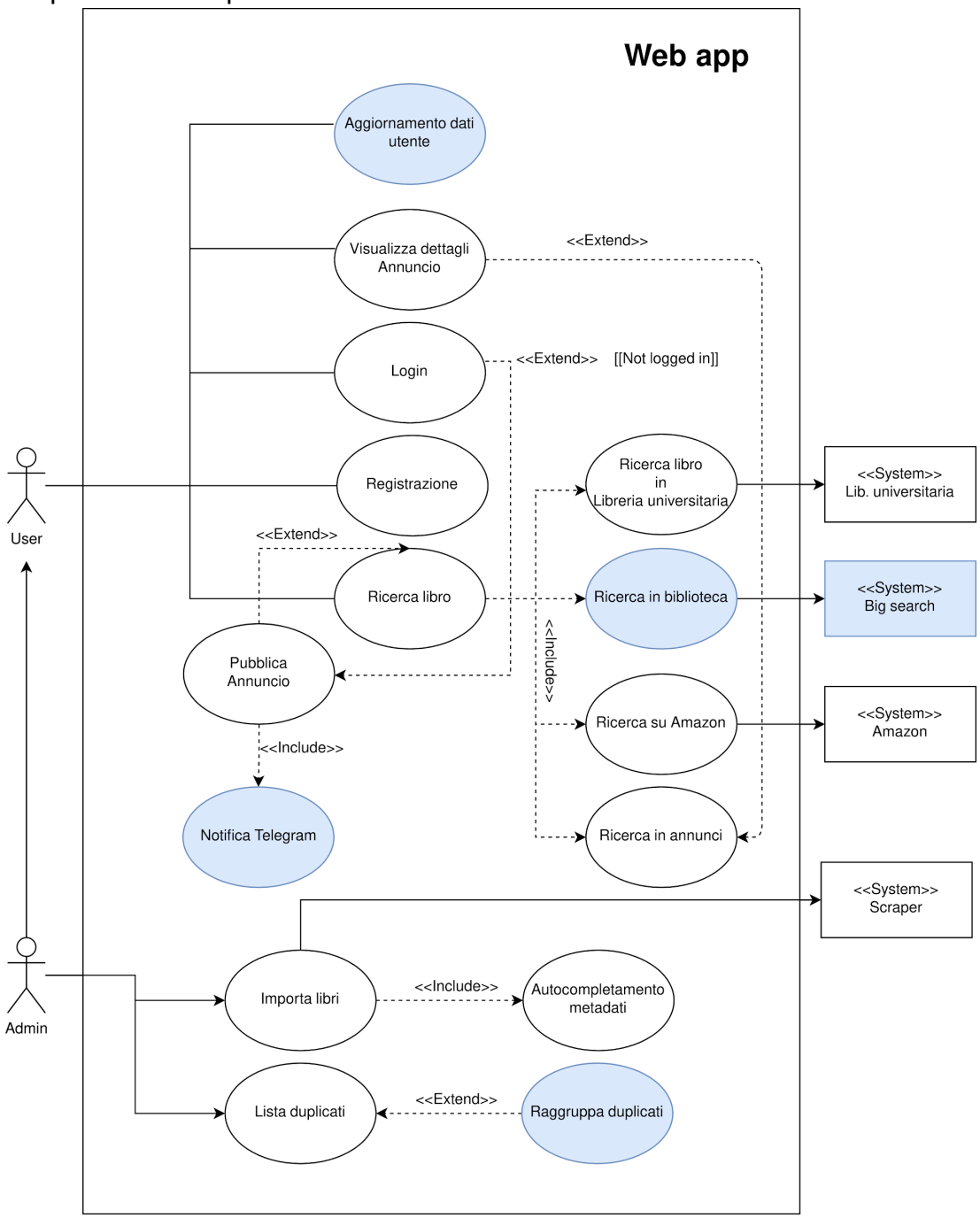
Iterazione 2

Nuovi casi d’uso

Per l’iterazione 1 abbiamo deciso di implementare i seguenti nuovi casi d’uso:

- Aggiornamento dei dati utente
- Ricerca di un libro nel sistema bibliotecario
- Invio di una notifica sul canale Telegram alla pubblicazione di un annuncio
- Ricerca e raggruppamento duplicati

Dal diagramma sottostante è possibile analizzare l’evolversi dello stato del progetto rispetto alla iterazione precedente, in particolare si può notare i nuovi casi d'uso implementati di colore azzurrino, mentre la restante parte non evidenziata rispecchia l’architettura già in produzione prima della implementazione delle nuove features.



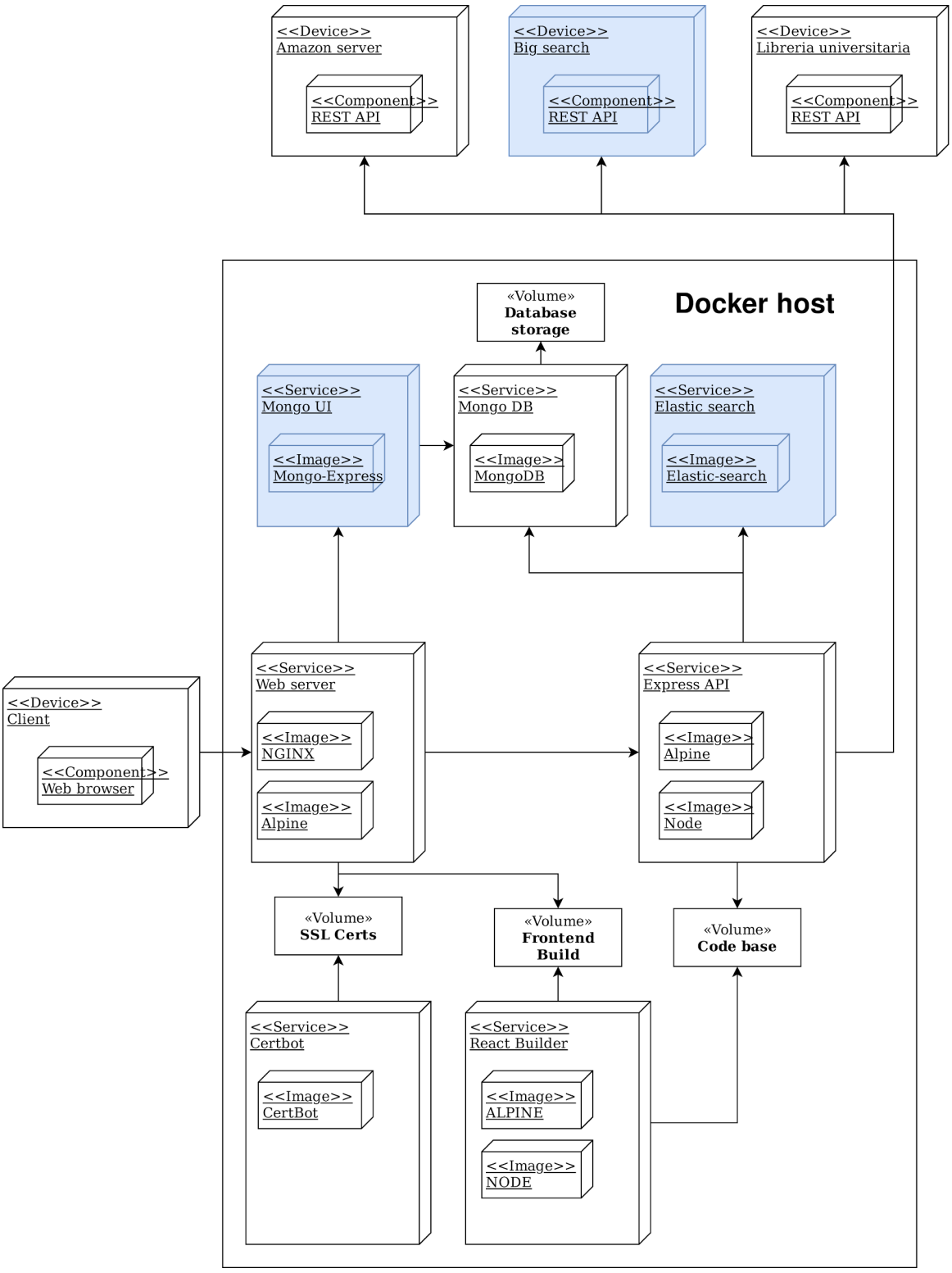
Nuova architettura

Deployment diagram avanzato

Anche nell'iterazione 2 è stata migliorata ulteriormente l'architettura del sistema, andando ad aggiungere i componenti evidenziati in azzurro nello schema che segue.

Nel dettaglio, sono stati aggiunti i seguenti servizi:

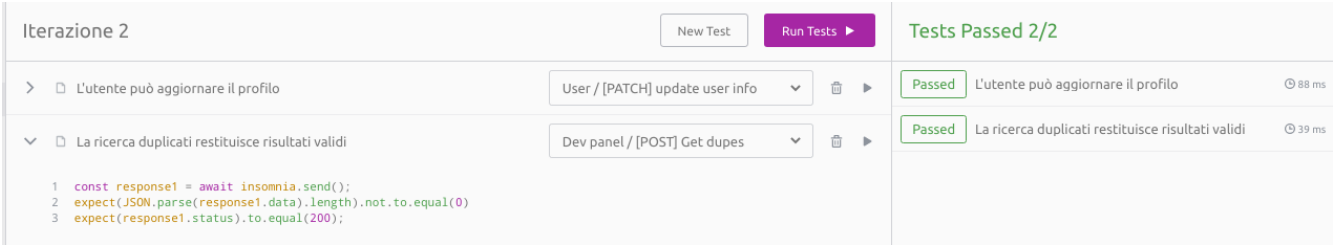
- MongoUI: Questo container espone un'interfaccia web che consente di amministrare il database per agevolare un'eventuale diagnosi di problemi dell'applicazione e monitorare lo stato dei record in produzione.
- Elasticsearch: Questo container contiene un database particolarmente flessibile che permette di effettuare ricerche full-text. Questo ha consentito di implementare l'autocompletamento nella barra di ricerca del sito: se l'utente inserisce un titolo incompleto o con errori di battitura il sistema è in grado di ovviare al problema e fornire risultati pertinenti.
- Interfacciamento con BigSearch: Il sistema si interfaccia con il server bigSearch in modo da ottenere dati aggiornati riguardo il catalogo di libri disponibili in biblioteca, fornendo all'utente un'esperienza più completa.


<<Device>>
Amazon server
<<Component>>
REST API
<<Device>>
Big search
<<Component>>
REST API
<<Device>>
Libreria universitaria
<<Component>>
REST API
Docker host
«Volume»
Database
storage
<<Service>>
Mongo UI
<<Image>>
Mongo-Express
<<Service>>
Mongo DB
<<Image>>
MongoDB
<<Service>>
Elastic search
<<Image>>
Elastic-search
<<Device>>
Client
<<Component>>
Web browser
<<Service>>
Web server
<<Image>>
NGINX
<<Image>>
Alpine
<<Service>>
Express API
<<Image>>
Alpine
<<Image>>
Node
«Volume»
SSL Certs
«Volume»
Frontend
Build
«Volume»
Code base
<<Service>>
Certbot
<<Image>>
CertBot
<<Service>>
React Builder
<<Image>>
ALPINE
<<Image>>
NODE

Test

Sono stati aggiunti i seguenti test:

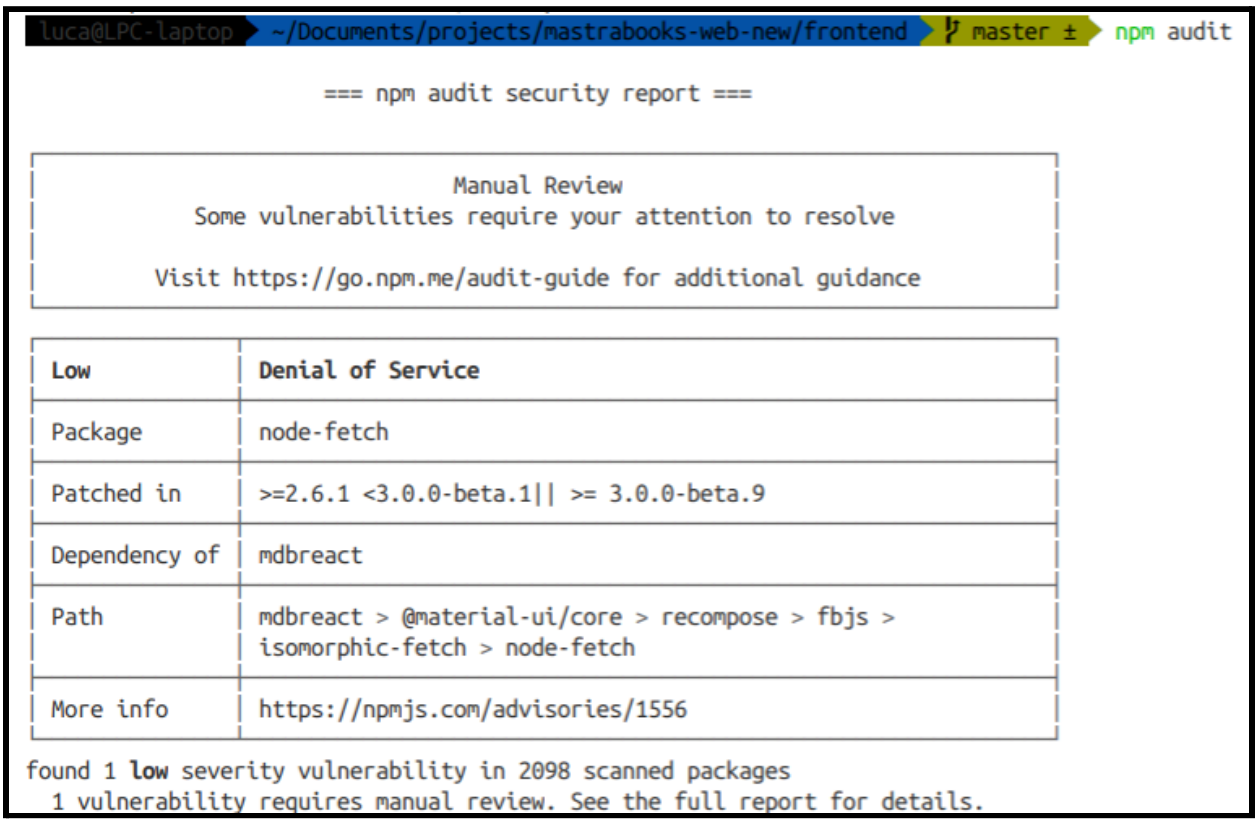
- Aggiornamento dati del profilo: il test controlla che l'utente possa inserire nuovi dati aggiornati e che i dati vengano aggiornati con successo dall'API
- Ricerca duplicati: Il test controlla che l'API fornisca un numero minimo di "duplicati" a seguito di una soglia di threshold alta, che garantisce l'apparizione di "falsi positivi" che in questo caso sono però utili a verificare che l'algoritmo fornisca risultati.



Lo screenshot mostra l'esito (PASSED) dei due test aggiunti

Viene ora riportato il report dell'analisi statica del progetto, eseguito con npm audit, nella quale viene mostrato che è stata rilevata una sola vulnerabilità classificata come rischio basso.

Non è stato possibile per noi risolvere questa vulnerabilità in quanto non dipendeva dal nostro codice, ma da una libreria in uso nel nostro progetto.



Pseudocodice parte algoritmica

#Algoritmo Inserimento Score

Algoritmo addScoreNodeRecursive(root, newElem)

if (newElem.score >= root.score)

//go to right

if (root.right != null)

addScoreNodeRecursive(root.right, newElem)

else

root.right = newElem

else

//go to left

if (root.left != null)

addScoreNodeRecursive(root.left, newElem)

```
    else
        root.left = newElem;
```

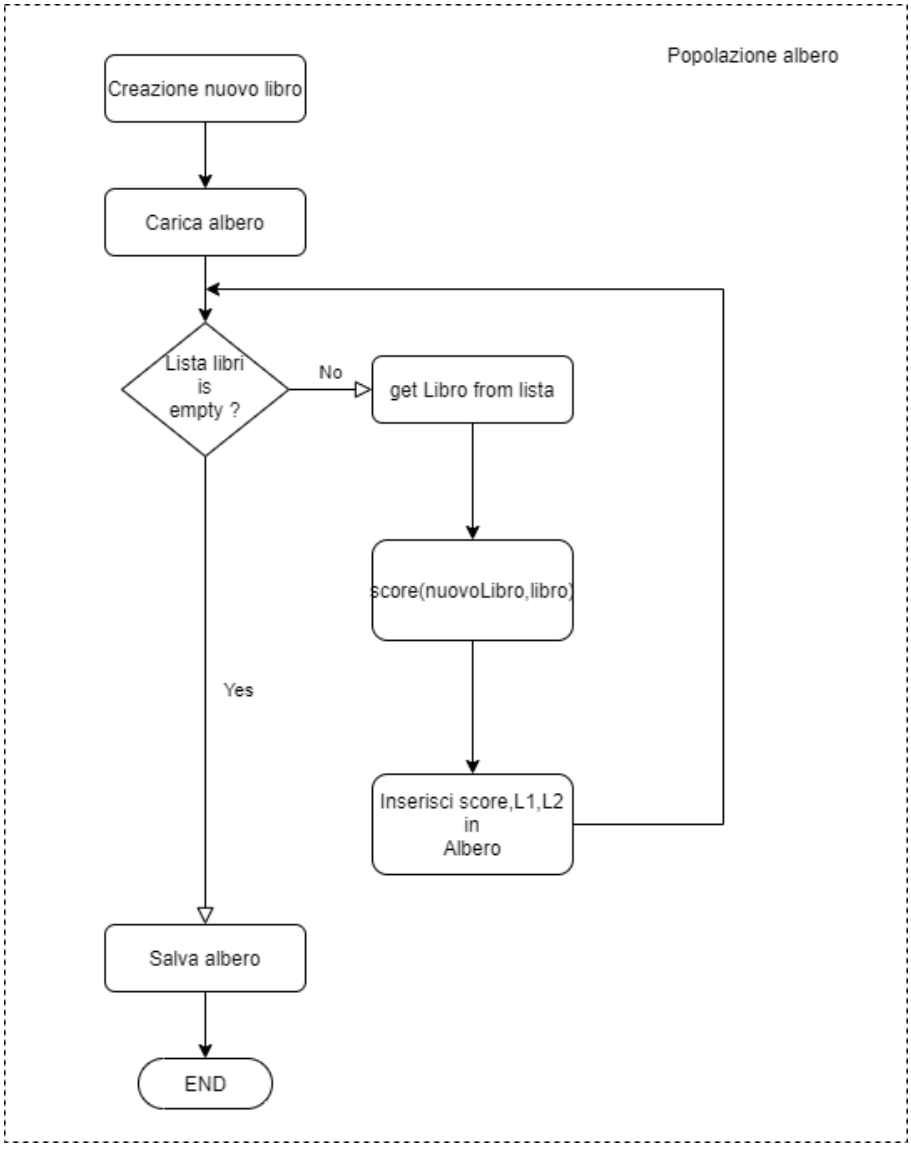
#Algoritmo stampa in base al threshold definito

```
Algoritmo printBasedOnThresholdRecursive(root, threshold, output)
    if (root != null)
        if (threshold < root.score)
            printBasedOnThresholdRecursive(root.left, threshold, output)
        else
            //stampo tutto quello che c'e a sinistra
            printTreeRecursive(root.left, output)
            //stampo il nodo corrente
            output.push(root.data)
            //analizzo il ramo destro
            printBasedOnThresholdRecursive(root.right, threshold, output);
```

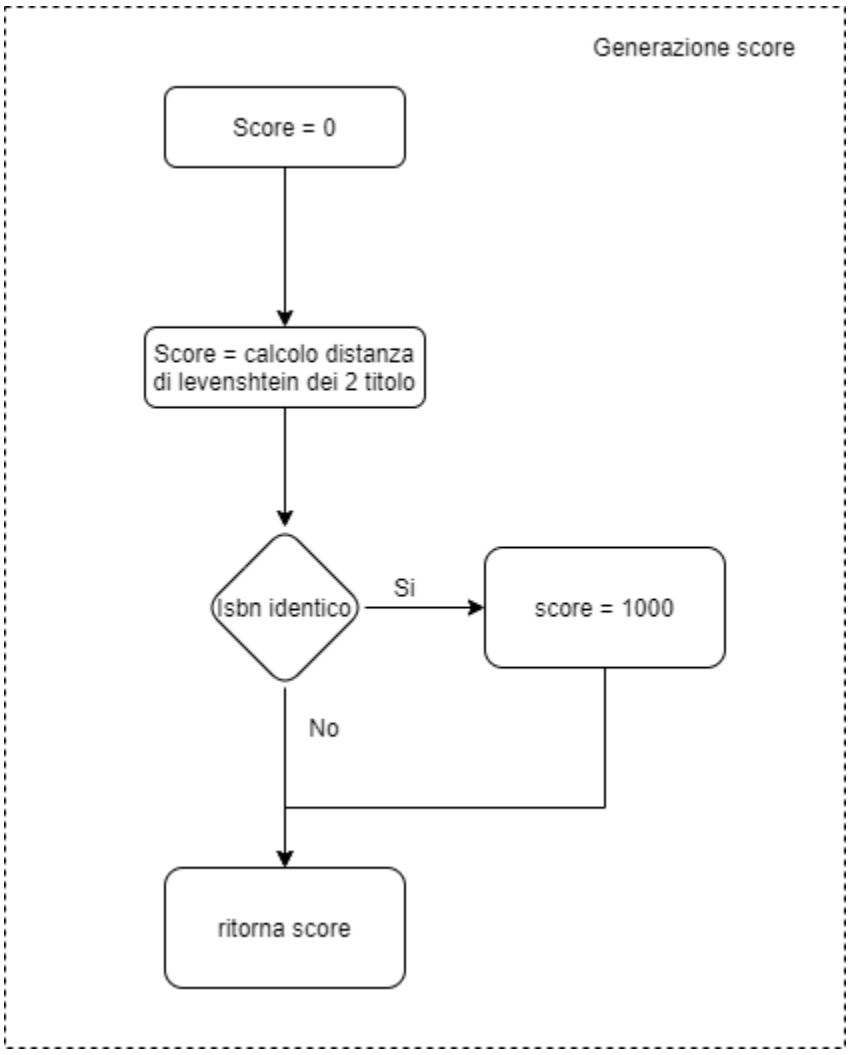
Flowchart

Per maggior chiarezza abbiamo realizzato anche i flowchart della parti algoritmiche implementate.

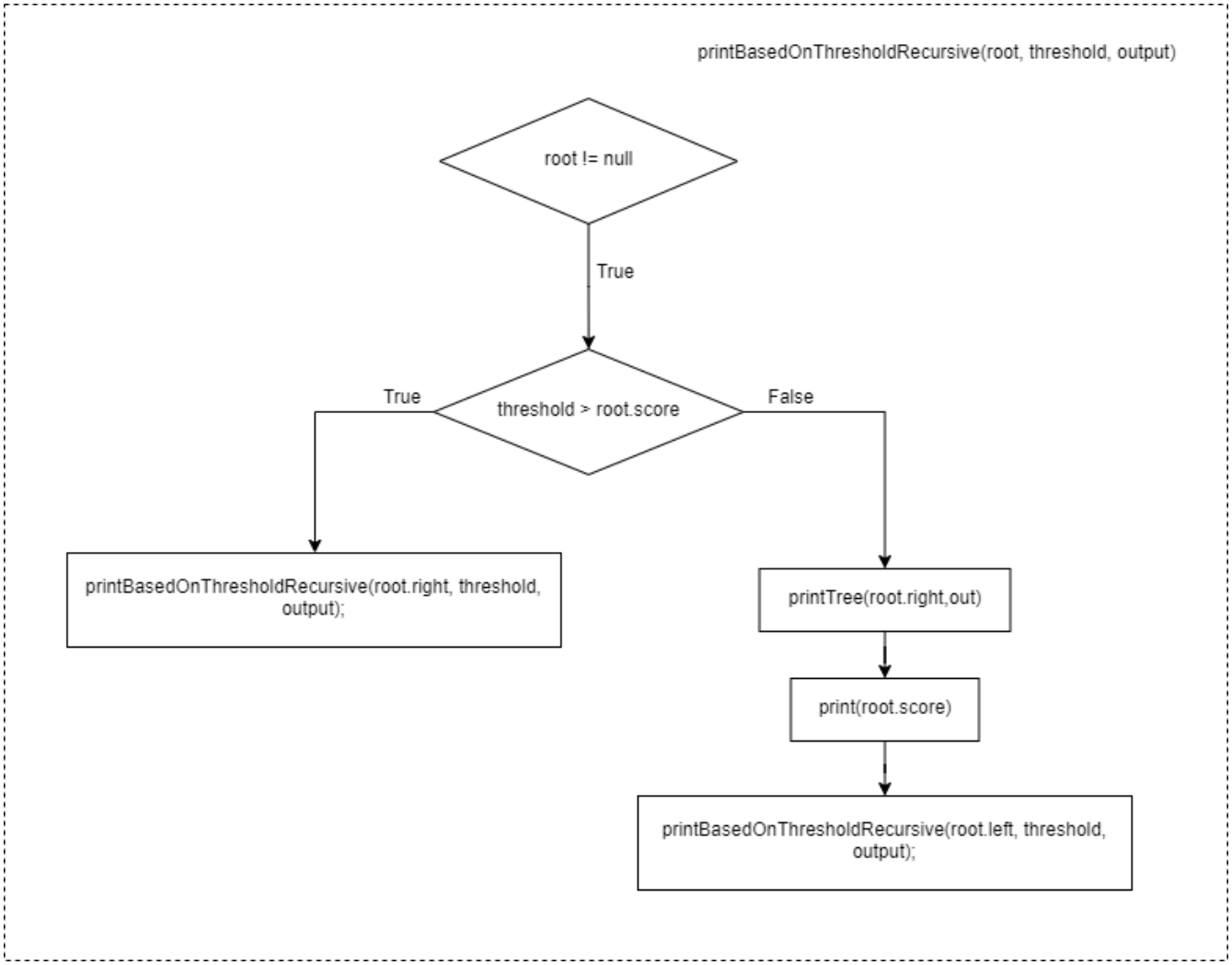
Popolazione albero



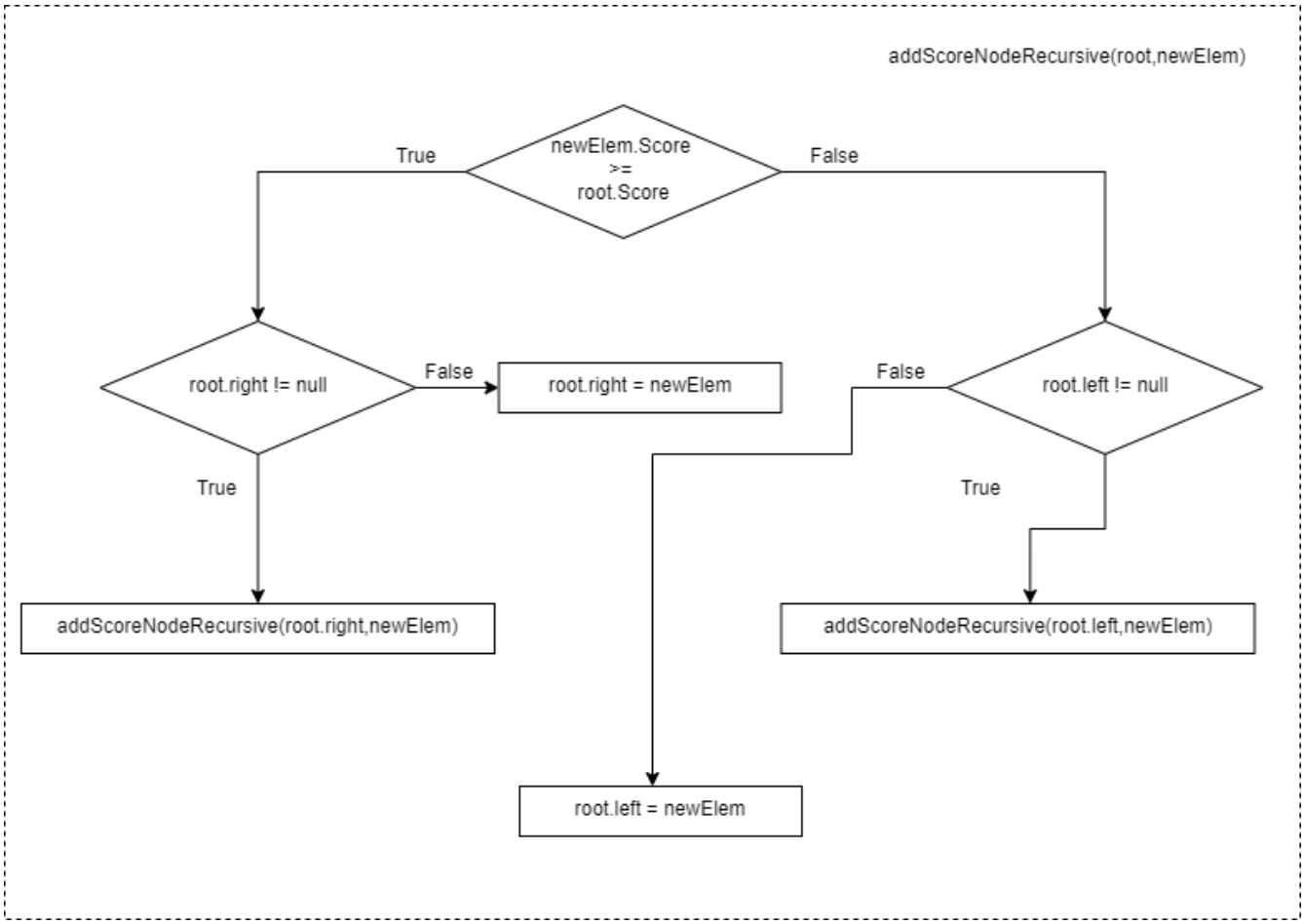
Generazione Score



Ricerca duplicati



Aggiungi nodo



Viste prodotte

Home page

Si tratta della pagina principale del sito, vengono fornite all'utente le principali funzionalità implementate nella nostra applicazione.

Tra cui:

- Ricerca Libro (generica)
- Carosello libri di maggior interesse
- Accesso alla pagina di Login/Logout, registrazione e profilo

Seguono le pagine di “visualizzazione libro”, che espone i restanti casi d’uso riguardanti le operazioni possibili sui libri, e “profilo utente”, che permette le operazioni di visualizzazione/modifica e rimozione di annunci pubblicati.

MastraBooks

Struttura e progetto dei calcolatori. Con e-book

di David A. Patterson, John L. Hennessy

Zanichelli

9788808352026

Visualizza su Amazon

Vendi il tuo usato

Disponibilità usato

Sottolineato a matita
di luca.rillosi
€ 25.00

Vendi il tuo usato

Disponibilità nuovo

Visualizza su Amazon

Visualizza su Libreria Universitaria

Made in UniBg with ❤ and a lot of 🍩

mastrabooks@gmail.com

MastraBooks

Home

Profilo

Logout

Account

Email: luca.rillosi@gmail.com
Username: luca.rillosi

I miei annunci

Titolo	Prezzo
Analisi matematica 1	€ 21.00
Elementi di fisica vol.1	€ 19.00
Dai fondamenti agli oggetti. Corso di programmazione Java. Con aggiornamento online	€ 22.00
Sistemi operativi. Concetti ed esempi	€ 29.00
Struttura e progetto dei calcolatori. Con e-book	€ 25.00
Elementi di fisica vol.2	€ 22.00

Copyright © MastraNetwork 2021

Made in UniBg with ❤ and a lot of 🍩

mastrabooks@gmail.com

Manuale di installazione server e primo avvio

Le seguenti istruzioni forniscono la procedura di installazione su un server con Ubuntu Server 20.04.
NB: è necessario disporre di un account con permessi di amministrazione (root).

Installazione git

Dalla tua shell, installa Git usando il comando apt-get:

- *sudo apt-get update $ sudo apt-get install git*

Configura il tuo nome utente e e-mail Git utilizzando i seguenti comandi, sostituendo il nome di “Emma” con il tuo. Questi dettagli verranno associati ad ogni commit creata

- *git config --global user.name "Emma Paris" $ git config --global user.email "eparis@atlassian.com"*

Installazione docker e docker-compose

Aggiornare il gestore dei pacchetti apt e installare l'ultima versione di Docker Engine

- *sudo apt-get update*
- *sudo apt-get install docker-ce docker-ce-cli containerd.io*

Verifica che Docker Engine sia installato correttamente eseguendo l'immagine hello-world.

- *sudo docker run hello-world*

Il seguente comando scaricherà la versione 1.27.4 e salverà il file eseguibile in /usr/local/bin/docker-compose, e renderà questo software accessibile globalmente come docker-compose.

- *sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose*

Quindi, imposta le autorizzazioni corrette in modo che il comando docker-compose possa essere eseguito correttamente

- *sudo chmod +x /usr/local/bin/docker-compose*

Per verificare che l'installazione sia andata a buon fine, è possibile eseguire:

- *docker-compose --version*

Setup del progetto

Scaricare la repository attraverso il comando git clone come mostrato:

- *git clone git@gitlab.com:progetti-info3/mastrabooks-web-new.git*

Una volta scaricato codice sorgente del progetto, il setup si divide in 2 branche differenti a seconda dell'ambiente di sviluppo target.
In particolar modo si distingue l'environment di development dall''environnement di production.

Development set-up

1. Avvio

docker-compose up --build

Attendere fino a che non si è inizializzato tutto ed Elasticsearch smette di fare l'indexing.

NB: Se il container di ES da problemi ad avviarsi o continua a riavviarsi, lanciare il seguente comando

sysctl -w vm.max_map_count=262144

2. Importare DB

- Installare mongodb-tools (Arch) o mongo-tools (Debian) o su CentOS:

yum --nogpgcheck localinstall mongodb-database-tools-rhel80-x86_64-100.1.1.rpm

- mantenendo attivo il container mongo, importa il dump:

cd backend/resources

mongorestore

- Forza il reindex dei libri per portare ES al passo con MongoDB richiamando l'endpoint /dev-panel/force-reindex

Production set-up

1. Imposta alias dcp

Anzichè docker-compose usare dcp, un alias che si comporta come docker-compose ma utilizzando i file .yml per la production.

Va definito nel file ~/.bashrc

alias dcp="docker-compose -f docker-compose.yml -f docker-compose.prod.yml"

2. Avviare prod usando gli script di installazione

Eseguire gli script nella cartella scripts nell'ordine indicato dal nome del file.

NB: Gli script vanno eseguiti da dentro cartella scripts

cd scripts

./01-create-dummy-certs.sh

./02-start-production.sh

./03-install-letsencrypt.sh

Per osservare i logs in tempo reale:

dcp logs -f

3. Importare DB

La procedura è la stessa di development, ma deve essere disponibile un server Elasticsearch hostato da qualche parte. Host, port e credenziali vanno nei file docker-compose.

Resettare i container

Per eliminare il permanent-storage di tutti i container utilizza (anche se già stoppati)

docker-compose down -v

Esportare dump DB

cd backend/resources

mongodump -d mastrabooks