

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SOFTWARE ARCHITECTURE DESIGN - A.A. 2022 - 23



Documento di progetto

Studenti:

- Luca Campanile M63/1166
luca.campanile@studenti.unina.it
- Salvatore Criscuolo M63/1143
luca.campanile@studenti.unina.it

Repository GitHub: <https://github.com/cruelsuerte/heritart>

12/09/2023

Indice

1. PROCESSO DI SVILUPPO

2. ANALISI E SPECIFICA DEI REQUISITI

2.1 Requisiti informali

2.2 Requisiti funzionali

2.3 Requisiti non funzionali

3. MODELLAZIONE DEI CASI D'USO

3.1 Attori e casi d'uso

3.2 Diagramma dei casi d'uso

4. DOCUMENTO DI VISIONE

5. MODELLO DI DOMINIO

6. SCENARI

7. DOCUMENTO DI ARCHITETTURA E DESIGN

7.1 Pattern MVC

7.2 Architettura REST

7.3 Spring Boot

7.4 Thymeleaf

7.5 MongoDB

8. MODELLO DEI DATI

9. MODELLO DI PROGETTO

9.1 Class Diagram

9.2 Package Diagram

9.3 Component Diagram

9.4 Deployment Diagram

9.5 Sequence Diagram

10. TESTING FUNZIONALE

11. ISTRUZIONI PER L'INSTALLAZIONE

11.1 Creazione dell'immagine Docker

11.2 Installazione del container in locale

11.3 Deploy su Microsoft Azure

1. Processo di sviluppo

Per la realizzazione del progetto è stato adottato un processo di sviluppo UP con approccio agile, combinando gli aspetti formali previsti dal suddetto metodo iterativo con alcune pratiche tipiche delle metodologie agili come:

- Adattabilità e flessibilità: il progetto è pensato per adeguarsi facilmente alla correzione o all'aggiunta di nuovi requisiti, sia in fase di sviluppo che dopo il rilascio;
- Iterazioni di durata breve (1-2 settimane), che mirano al completamento rapido e incrementale di nuove funzionalità;
- Pair-programming e collaborazione: lo sviluppo ha richiesto la programmazione in coppia supportata dalla continua comunicazione tra i membri del team, promuovendo la condivisione di idee, decisioni e responsabilità.

Il processo di sviluppo, iniziato il 17/07/2023, ha previsto le seguenti fasi progettuali.

- **Ideazione** (dal 17/07 al 23/07):
 - Valutazione degli obiettivi di progetto;
 - Analisi e specifica dei requisiti;
 - Modellazione dei casi d'uso da implementare;
 - Produzione di un documento di visione.
- **Elaborazione** (dal 24/07 al 31/07):
 - Produzione dei seguenti elaborati, completati nella fase di costruzione:
 - **Modello di dominio**;
 - Definizione degli **scenari**;
 - **Documento di Architettura e Design**,
che descrive le scelte architettoniche adottate.
 - **Modello dei dati**;
 - **Modello di Progetto**, che include Class Diagram, Package Diagram, Component Diagram, Deployment Diagram, Sequence Diagram;
 - Implementazione della struttura basilare dell'applicazione,
con l'integrazione dei principali supporti e componenti software;
 - Pianificazione delle iterazioni da condurre nella fase di costruzione.

- **Costruzione** (dal 01/08 al 29/08)

Implementazione dell'applicazione organizzata in 4 iterazioni settimanali

Iterazione	Funzionalità prodotte	Casi d'uso	Componenti software introdotte
#1	Registrazione; Gestione dei token e invio delle mail di conferma; Autenticazione e controllo degli accessi basato su ruolo.	Registrazione	Model: Utente, Verification Token; View: login.html; DAO: UtentiRepository, TokenRepository; Controller: ControllerAccess; Servizi: HeritartConfig, AuthenticationService, MailSender
#2	Funzionalità e interfacce utente riservate al gestore, che consentono la creazione di opere e aste; Validazione degli input e gestione degli errori;	Aggiungi Opera; Aggiungi Asta	Model: Opera, Asta; View: aggiungi_opera.html, aggiungi_asta.html; DAO: OpereRepository, AsteRepository; Controller: ControllerGestore, ErrorHandler;
#3	Funzionalità e interfacce utente riservate sia al cliente che al gestore, che consentono la visualizzazione del catalogo, la ricerca, la visualizzazione delle opere di un'asta.	Ricerca; Visualizza Asta	View: home.html, opera.html, asta.html; Controller: ControllerCatalog, ControllerAsta
#4	Funzionalità e interfacce utente riservate esclusivamente al cliente, ovvero l'invio di un'offerta per un'opera all'asta, la visualizzazione della collezione di opere aggiudicate; Verifica transazionale delle offerte; Gestione automatica delle decorrenze relative ad aste e token.	Offerta; Visualizza Collezione; Gestisci aste	Model: Offerta; View: collection.html; DAO: OfferteRepository; Controller: ControllerCliente; Servizi: TransactionService, ExpirationSolver

- **Transizione** (dal 30/08 al 07/09)

Testing, revisione del progetto, rilascio dell'applicazione.

2. Analisi e specifica dei requisiti

2.1 Requisiti informali

Si vuole realizzare un'applicazione web che gestisca un catalogo di aste per la compravendita di opere d'arte.

Un'asta prevede una data di inizio e di fine e consiste in una raccolta di opere in vendita.

Un'opera d'arte (lotto) può essere un dipinto o una scultura.

Ad ogni opera è associato un titolo, l'artista, la data di creazione, la tecnica adottata, la corrente artistica (genere), le dimensioni, il materiale, lo stato di conservazione, una descrizione e informazione relative alla proprietà e alla provenienza.

Un lotto in vendita può prevedere un set di foto da esporre e una base d'asta.

L'applicazione prevede due tipologie di utenti: cliente e gestore.

Un utente deve registrarsi al portale indicando nome, e-mail e, nel caso sia un cliente, informazioni di contatto. Gli utenti possono ultimare la registrazione accedendo ad un link presente in una mail di conferma.

Un gestore si occupa dell'inserimento di nuove opere o aste al catalogo, riportando tutte le informazioni necessarie.

Gli utenti possono visualizzare tutte le aste presenti nel catalogo e ricercare un'asta di interesse indicando lo stato dell'asta (in programma o in corso), un titolo, un artista o un genere di riferimento.

Gli utenti possono visualizzare le opere messe in vendita in un'asta e, nel caso in cui l'asta sia in corso o terminata, accedere ad una pagina che riporta nel dettaglio le informazioni relative ad un'opera, tra cui le ultime migliori offerte presentate.

Nella stessa pagina, un cliente può presentare delle offerte finché l'asta è in corso.

Un cliente può inviare un'offerta incrementando quella minima di un importo fisso (50 €), a meno che non abbia già presentato l'ultima migliore offerta. L'offerta minima dipende dalla base d'asta o dall'ultima migliore offerta presentata. Un lotto sarà aggiudicato al cliente che ha presentato la migliore offerta prima del termine dell'asta. Tale acquirente sarà avvisato in modo automatico tramite mail. Nel caso in cui non siano pervenute offerte per un'opera, essa potrà nuovamente essere inserita in un'asta.

Un cliente può inoltre accedere alla personale collezione di opere aggiudicate.

L'applicazione gestisce autonomamente l'inizio e il termine delle aste.

2.2 Requisiti funzionali

È stata ricavata una definizione chiara e dettagliata dei requisiti previsti dall'applicazione:

1. L'applicazione si occupa della gestione di un catalogo di aste di opere d'arte;
2. L'applicazione deve fornire agli utenti la possibilità di registrarsi e di confermare la registrazione in seguito alla ricezione di una mail;
3. Gli utenti con un account non attivo possono chiedere il rinvio della mail di conferma;
4. Gli utenti possono avere il ruolo di cliente o di gestore;
5. Gli utenti registrati sono caratterizzati da e-mail, password, nome e cognome e, nel caso in cui si tratti di un cliente, da informazioni di contatto come residenza e recapito telefonico;
6. L'applicazione deve fornire agli utenti registrati la possibilità di autenticarsi e controllare l'accesso alle pagine e alle funzionalità sulla base del ruolo;
7. L'applicazione deve fornire al gestore delle funzionalità per l'inserimento di nuove opere o aste al catalogo;
8. Un'opera inserita può essere un dipinto o una scultura;
9. Un'opera è caratterizzata dai seguenti campi obbligatori: titolo, artista, tipologia, data di creazione, stato di conservazione, proprietà e provenienza;
10. Un'opera è caratterizzata dai seguenti campi facoltativi: descrizione, dimensioni, corrente artistica, tecnica adottata, materiale (nel caso sia una scultura), set di foto, base d'asta.
11. Un'asta è caratterizzata dai seguenti campi obbligatori: titolo, data d'inizio, data di fine, set di opere disponibili alla vendita;
12. Un'asta è caratterizzata dai seguenti campi facoltativi: descrizione, foto;
13. L'applicazione deve impedire l'inserimento di aste con date già trascorse e accettare una data di fine superiore di almeno 3 ore a quella di inizio;
14. L'applicazione deve prevedere la validazione dei dati inseriti sia front-end che back-end e fornire opportuni messaggi di errore;
15. L'applicazione deve fornire agli utenti autenticati la possibilità di visualizzare le aste del catalogo e ricercare un'asta di interesse indicando lo stato dell'asta (in programma o in corso), un titolo, un artista o un genere di riferimento;
16. L'applicazione deve fornire agli utenti autenticati la possibilità di visualizzare le opere di un'asta;
17. Nel caso in cui un'asta sia in corso o terminata, l'applicazione deve fornire agli utenti autenticati la possibilità di accedere ad una pagina che riporta nel dettaglio le informazioni relative ad un'opera all'asta, tra cui le ultime migliori offerte presentate;

18. Nel caso in cui un'asta sia in corso, l'applicazione deve fornire al cliente la funzionalità di invio di un'offerta per un'opera all'asta;
19. Un cliente non può inviare una nuova offerta se ha già presentato l'ultima migliore offerta per un'opera;
20. Un cliente può presentare un'offerta solo se è pari o superiore alla minima offerta presentabile;
21. Un cliente può inserire un'offerta aumentando quella minima di un incremento fisso (50 €);
22. L'offerta minima è pari alla base d'asta o all'ultima migliore offerta + 50 €;
23. L'applicazione deve gestire opportunamente la verifica e il possibile invio di offerte in concomitanza;
24. Nel caso in cui un'asta sia terminata, un'opera è aggiudicata al cliente che ha presentato l'ultima migliore offerta entro il termine dell'asta;
25. L'applicazione prevede l'invio automatico di una mail di avviso al cliente che ha presentato la migliore offerta per un'opera quando un'asta è terminata;
26. L'applicazione deve fornire ad un cliente la possibilità di accedere alla personale collezione di opere aggiudicate;
27. Nel caso in cui un'asta sia terminata, un'opera che non ha ricevuto offerte torna disponibile all'inserimento in nuove aste;
28. L'applicazione deve gestire autonomamente le operazioni che riguardano l'inizio e il termine di un'asta;

2.3 Requisiti non funzionali

L'applicazione deve soddisfare i seguenti requisiti non funzionali:

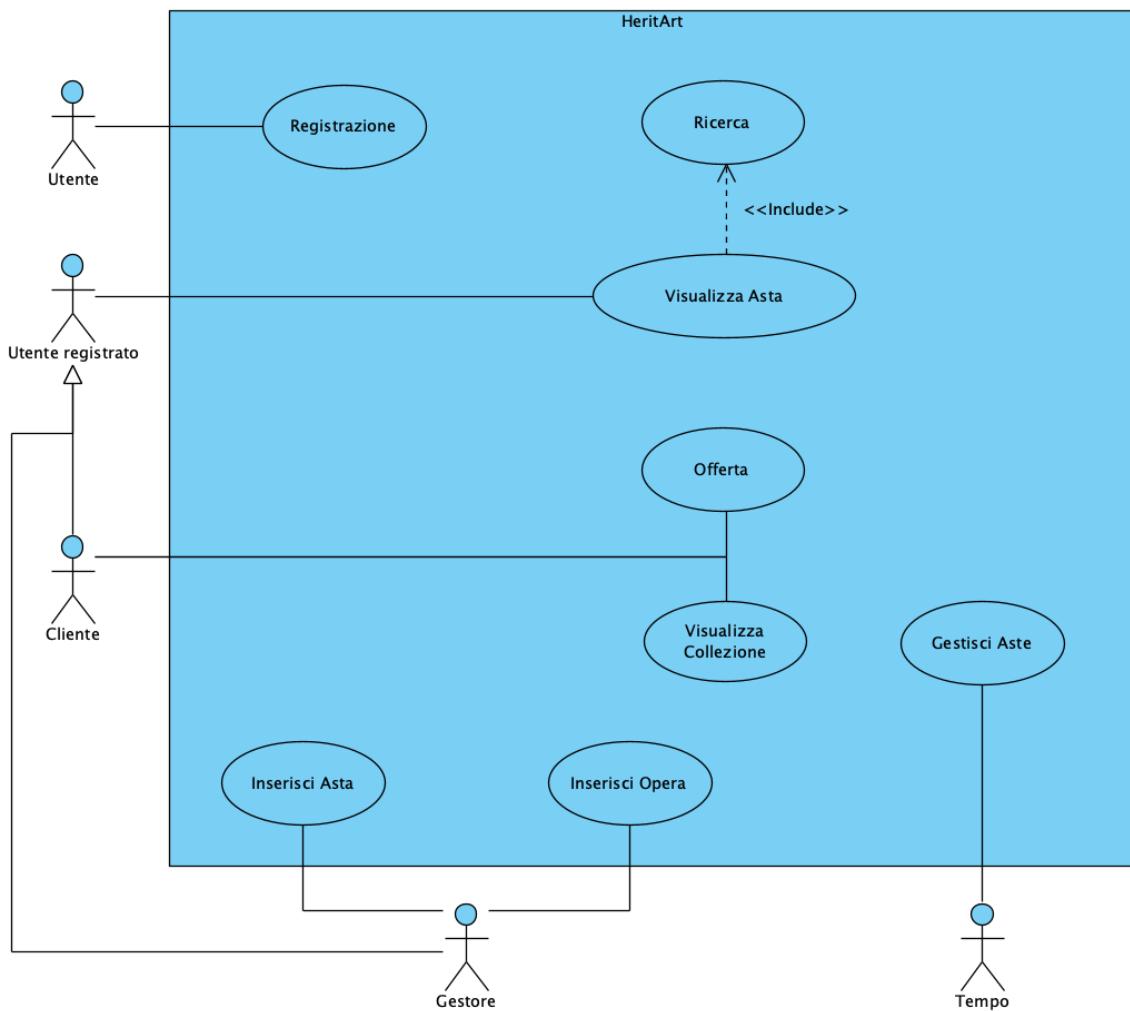
- *Usabilità*: l'interazione con l'interfaccia utente deve essere intuitiva e semplice;
- *Portabilità* e facilità di deploy;
- *Responsiveness* e compatibilità con diversi dispositivi e web browser;
- *Robustezza*: l'applicazione deve comportarsi in maniera accettabile anche in seguito all'invio di input non validi, fornendo opportuni e chiari messaggi di errore;
- *Scalabilità architetturale*: la struttura applicativa deve consentire facilmente l'introduzione di nuove funzionalità;
- *Sicurezza* (autenticazione e autorizzazione): l'applicazione deve garantire che l'accesso alle risorse sia consentito solo ad utenti autorizzati e in precise circostanze.

3. Modellazione dei casi d'uso

3.1 Attori e casi d'uso

Caso d'uso	Attori Primari
Registrazione	Utente
Ricerca	Utente Registrato
Visualizza Asta	Utente Registrato
Offerta	Cliente
Visualizza Collezione	Cliente
Aggiungi Opera	Gestore
Aggiungi Asta	Gestore
Gestisci Aste	Tempo

3.2 Diagramma dei casi d'uso



4. Documento di visione

L'applicazione prevede un controllo degli accessi basato sui ruoli degli utenti (CLIENTE e GESTORE) e sullo stato delle aste in funzione del tempo. In seguito alla registrazione, all'utente è inviata un'email di conferma con un link il cui accesso consente l'attivazione definitiva dell'account. Tale link include un **Verification Token** dalla durata di un giorno. L'autenticazione non è consentita agli utenti con un account non attivo, che tuttavia possono chiedere il rinvio della mail di conferma.

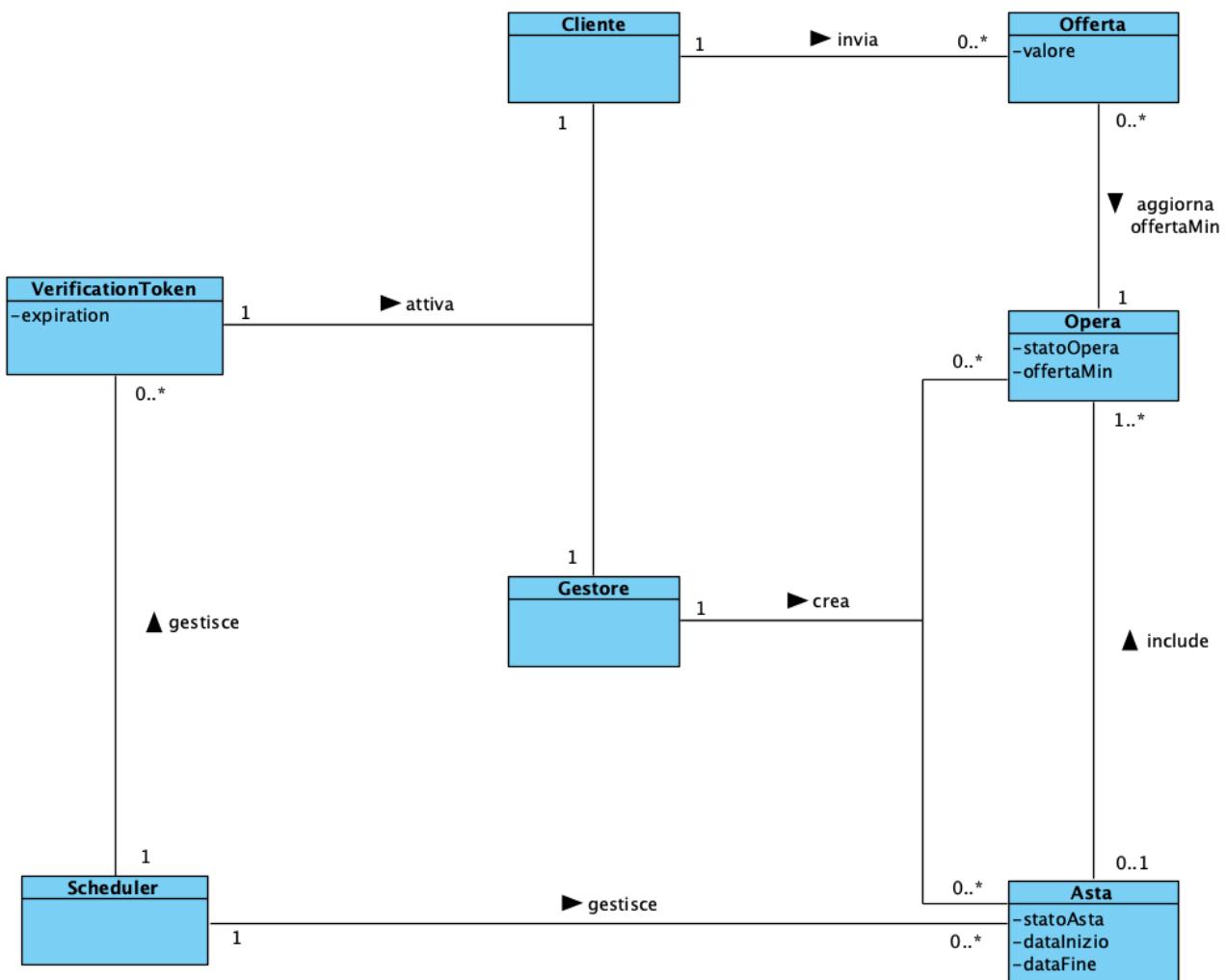
Entrambe le tipologie di utenti possono accedere al **catalogo** delle aste ed effettuare una **ricerca** basata sullo stato delle aste o su un titolo, un artista o un genere di riferimento. Gli utenti possono accedere alla sezione che mostra la raccolta di opere di un'asta. Nel caso in cui l'asta sia in corso o terminata, è attivo un bottone che consente di accedere alla pagina di dettaglio di ciascun'opera. In questa pagina è possibile visualizzare le informazioni dell'opera e le **ultime migliori offerte** presentate. Finche l'asta è in corso, un cliente ha la possibilità di presentare offerte per l'opera di interesse. Il cliente non può inviare un'offerta che sia inferiore all'offerta minima indicata o nel caso in cui abbia già presentato l'ultima migliore offerta. Le verifiche e le operazioni relative all'invio di un'offerta avvengono come **unica transazione**, in modo da evitare possibili conflitti in caso di richieste concomitanti.

Un gestore può accedere alla sezione che consente l'inserimento di nuove opere o aste. La creazione di un'opera richiede, tra le varie opzioni, la specifica della tipologia (DIPINTO o SCULTURA), delle condizioni di conservazione, l'eventuale aggiunta di un set foto o di una base d'asta (0 euro come valore predefinito). Un'opera appena inserita risulta DISPONIBILE e può essere messa all'ASTA da un gestore. Un'opera all'asta viene AGGIUDICATA quando, al termine dell'asta, almeno un'offerta è stata presentata. Un'email di avviso è inviata al miglior offerente. Un cliente può inoltre accedere ad una sezione, chiamata 'myCollection', che riporta la personale collezione di opere aggiudicate. Nel caso in cui un'opera non abbia ricevuto offerte durante un'asta, essa ritorna ad essere DISPONIBILE.

La creazione di un'asta prevede l'inserimento di una data di inizio e di fine e la scelta di una o più opere disponibili. Un'asta appena inserita risulta PROGRAMMATA e diventa IN CORSO al presentarsi della data di inizio o TERMINATA al presentarsi della data di fine. La gestione delle aste e dei token è automatizzata mediante uno **Scheduler**, che verifica le decorrenze ed effettua tutte le operazioni che ne scaturiscono.

5. Modello di dominio

Il modello di dominio fornisce una vista concettuale delle relazioni tra le principali entità del dominio applicativo.



6. Scenari

Registrazione

Caso d'uso:	Registrazione
Attore primario	Utente
Attore secondario	
Descrizione	Un utente inserisce le informazioni necessarie per richiedere l'attivazione di un account
Pre-Condizioni	
Sequenza di eventi principale	<ol style="list-style-type: none">1. L'utente seleziona "Registrati"2. L'utente inserisce un username univoco, e-mail, password e informazioni di contatto3. L'utente seleziona "Conferma"4. Se i dati inseriti sono corretti<ol style="list-style-type: none">4.1. Il sistema associa un account all'utente e invia una mail di conferma4.2. Se l'utente conferma la registrazione<ol style="list-style-type: none">4.2.1. Il sistema attiva l'account associato all'utente5. Altrimenti<ol style="list-style-type: none">5.1. Il sistema restituisce un errore su uno o più campi non validi
Post-Condizioni	L'utente è registrato al portale
Casi d'uso correlati	nessuno
Sequenza di eventi alternativi	Nessuna

Ricerca

Caso d'uso:	Ricerca
Attore primario	Utente registrato
Attore secondario	
Descrizione	L'utente ricerca un'asta di interesse
Pre-Condizioni	L'utente registrato ha effettuato l'autenticazione. Il sistema mostra la lista di aste più recenti.
Sequenza di eventi principale	<ol style="list-style-type: none">1. L'utente inserisce il titolo di un'asta o di un'opera, un artista di interesse, un genere o lo stato dell'asta (tutte, in programma, in corso)2. L'utente seleziona "Ricerca"3. Se sono presenti aste nello stato indicato o con il titolo inserito o con opere appartenenti al genere o all'artista riportati<ol style="list-style-type: none">3.1. Il sistema mostra la lista di aste con le caratteristiche ricercate4. Altrimenti<ol style="list-style-type: none">4.1. Il sistema restituisce il messaggio "Nessun risultato trovato."
Post-Condizioni	
Casi d'uso correlati	nessuno
Sequenza di eventi alternativi	Nessuna

Visualizza Asta

Caso d'uso:	Visualizza Asta
Attore primario	Utente registrato
Attore secondario	
Descrizione	L'utente accede ad un'asta in corso o terminata
Pre-Condizioni	L'utente registrato ha effettuato l'autenticazione.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Include (Ricerca) 2. L'utente sceglie un'asta di interesse e seleziona "Accedi" 3. Il sistema mostra la lista delle opere messe all'asta 4. Se l'asta è in corso o è terminata <ol style="list-style-type: none"> 4.1. L'utente sceglie un'opera di interesse e seleziona "Asta" 4.2. Il sistema mostra le informazioni relative al lotto selezionato e le ultime migliori offerte
Post-Condizioni	
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

Offerta

Caso d'uso:	Offerta
Attore primario	Cliente
Attore secondario	
Descrizione	Il cliente presenta un'offerta per un'opera messa all'asta
Pre-Condizioni	L'asta è in corso e il cliente visualizza il dettaglio di un lotto in vendita
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente può presentare un'offerta aumentando quella migliore corrente di un incremento fisso 2. Se il cliente non ha già presentato l'ultima migliore offerta <ol style="list-style-type: none"> 2.1. Se l'offerta proposta è migliore di quella corrente <ol style="list-style-type: none"> 2.1.1. Il sistema restituisce il messaggio "Nuova offerta presentata con successo." 3. Altrimenti <ol style="list-style-type: none"> 3.1. Il sistema restituisce il messaggio "Hai già presentato l'ultima migliore offerta. Attendi una nuova offerta."
Post-Condizioni	Viene aggiornata la miglior offerta presentata per il lotto selezionato
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

Visualizza Collezione

Caso d'uso:	Visualizza Collezione
Attore primario	Gestore
Attore secondario	
Descrizione	Il cliente visualizza la collezione personale di opere aggiudicate
Pre-Condizioni	Il cliente ha effettuato l'autenticazione
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il cliente seleziona "MyCollection" 2. Se esiste almeno un'opera aggiudicata associata al cliente <ol style="list-style-type: none"> 2.1. Il sistema mostra la lista di opere aggiudicate 3. Altrimenti <ol style="list-style-type: none"> 3.1. Il sistema restituisce il messaggio "Nessun'opera aggiunta alla collezione."
Post-Condizioni	
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

Gestisci aste

Caso d'uso:	Gestisci aste
Attore primario	Tempo
Attore secondario	
Descrizione	Il sistema aggiorna lo stato delle aste quando decorrono le relative date di inizio e di fine
Pre-Condizioni	Il sistema prevede delle aste in programma o in corso
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Per ogni asta in programma <ol style="list-style-type: none"> 1.1. Se la data attuale è maggiore o uguale della data di inizio dell'asta <ol style="list-style-type: none"> 1.1.1. Il sistema aggiorna lo stato dell'asta a "In corso" 2. Per ogni asta in corso <ol style="list-style-type: none"> 2.1. Se la data attuale è maggiore o uguale della data di fine dell'asta <ol style="list-style-type: none"> 2.1.1. Il sistema aggiorna lo stato dell'asta a "Terminata" 2.1.2. Per ogni lotto dell'asta <ol style="list-style-type: none"> 2.1.2.1. Se esiste una migliore offerta presentata da un utente <ol style="list-style-type: none"> 2.1.2.1.1. Il sistema invia una mail all'utente comunicando la vittoria del lotto e gli estremi di pagamento. 2.1.2.1.2. Il sistema aggiorna lo stato dell'opera a "Aggiudicata" 2.1.2.2. Altrimenti <ol style="list-style-type: none"> 2.1.2.2.1. Il sistema aggiorna lo stato dell'opera a "Disponibile"
Post-Condizioni	Viene aggiornato lo stato delle aste e delle opere
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

Aggiungi Opera

Caso d'uso:	Aggiungi Opera
Attore primario	Gestore
Attore secondario	
Descrizione	Il gestore aggiunge un'opera da mettere all'asta
Pre-Condizioni	Il gestore ha effettuato l'autenticazione
Sequenza di eventi principale	<p>4. Il gestore seleziona "Aggiungi opera"</p> <p>5. Il gestore compila i campi relativi alle informazioni dell'opera</p> <p>6. Il gestore seleziona "Aggiungi"</p> <p>7. Se i dati inseriti non sono corretti</p> <p>7.1. Il sistema restituisce un errore su uno o più campi non validi</p> <p>8. Altrimenti</p> <p>8.1. Il sistema memorizza l'opera con lo stato "Disponibile"</p>
Post-Condizioni	Viene inserita l'opera nel sistema
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

Aggiungi Asta

Caso d'uso:	Aggiungi Asta
Attore primario	Gestore
Attore secondario	
Descrizione	Il gestore aggiunge un'asta in programma
Pre-Condizioni	Il gestore ha effettuato l'autenticazione
Sequenza di eventi principale	<p>1. Il gestore seleziona "Aggiungi asta"</p> <p>2. Il gestore compila i campi relativi alle informazioni dell'asta</p> <p>3. Il gestore seleziona almeno un'opera disponibile</p> <p>4. Se i dati indicati non sono corretti</p> <p>4.1. Il sistema restituisce un messaggio di errore</p> <p>5. Altrimenti</p> <p>5.1. Il sistema memorizza l'asta con lo stato "In programma"</p>
Post-Condizioni	Viene inserita l'asta nel sistema
Casi d'uso correlati	<i>nessuno</i>
Sequenza di eventi alternativi	Nessuna

7. Documento di Architettura e Design

7.1 Pattern MVC

Per il progetto proposto è stata scelta un'architettura di tipo client-server ed è stato adottato il pattern architettonico MVC (Model – View - Controller). Tale soluzione ben si presta alla realizzazione di un'applicazione web, garantendo la netta separazione tra la logica di presentazione e la logica di business. Il pattern è basato sulla separazione delle funzioni tra tre principali componenti software. Il **Model** si occupa della gestione dei dati rappresentativi del dominio dell'applicazione. La **View** consente la visualizzazione dei dati e l'interazione con essi da parte degli utenti. Il **Controller** traduce le richieste degli utenti, derivate dalla View, in operazioni relative ai dati del Model e stabilisce le pagine della View da presentare all'utente. È stato considerato un'ulteriore componente, denominato **DAO**, che funge da interfaccia verso il database, fornendo i metodi (query) che consentono di accedere ai dati persistenti interpretandoli come elementi del Model. La separazione dei componenti facilita particolarmente lo sviluppo modulare dell'applicazione, soprattutto quando è prevista una progettazione di tipo Object-Oriented.

7.2 Architettura REST

La logica di controllo è interamente realizzata in Java ed è resa ulteriormente scalabile mediante un approccio architettonico di tipo **REST**. Esso prevede che il Controller sia definito da un insieme di risorse web, univocamente identificate da URL. È possibile accedere alle risorse utilizzando metodi HTTP specifici (es. GET, POST) per il recupero o l'elaborazione di informazioni. I dati scambiati sono tipicamente rappresentati da documenti HTML o JSON. Tale procedura di comunicazione è nota come **API RESTful**. Un'architettura REST è anche “stateless”, poiché la richiesta di un client contiene tutte le informazioni necessarie per essere interpretata e gestita dal server. Mentre la logica operativa è completamente affidata al server, il client può farsi carico della presentazione dell'interfaccia utente (anche grazie all'esecuzione di codice Javascript) e del salvataggio di informazioni legate alla sessione.

7.3 Spring Boot

Lo sviluppo dell'applicazione ha previsto l'impiego del framework **Spring Boot**, che incorpora nativamente il pattern MVC e offre un supporto completo alla creazione di un'architettura REST. La caratteristica principale di Spring Boot è la configurazione automatica: applica le impostazioni e le dipendenze necessarie per l'applicazione in base alle convenzioni e alle librerie presenti nel progetto. Con lo strumento Maven è possibile integrare facilmente librerie esterne dichiarandole in un apposito file (pom.xml). Con i meccanismi di **Inversion of Control** (IoC) e **Dependency Injection** (DI), il framework si assume la responsabilità della creazione di specifici oggetti e delle loro dipendenze, che altrimenti andrebbero gestite in maniera esplicita. Le classi istanziate sono indicate da annotazioni speciali come `@Component`, `@Service`, `@Controller` o `@Repository`, a seconda del ruolo che svolgono all'interno dell'applicazione. Applicando la notazione `@Autowired` ad un'istanza di tali classi, Spring Boot ne ricava automaticamente un'implementazione con tutte le dipendenze necessarie, in modo da poterne sfruttare le funzionalità senza occuparsi della loro costruzione.

7.4 Thymeleaf

Thymeleaf è un generatore di template, supportato da Spring Boot, utilizzato per semplificare ulteriormente le interazioni tra i componenti del modello MVC. Esso consente al Controller di influenzare dinamicamente le pagine web presentate dalla View. Con Thymeleaf è possibile creare template HTML con tag ed espressioni speciali, che vengono interpretati ed elaborati lato server per adattare le pagine mostrate al client. Quando è richiesto l'accesso ad una pagina web, il Controller può contribuire alla generazione della pagina richiesta passando dati del Model o, in generale, del contesto applicativo.

Le espressioni Thymeleaf incorporate nei template utilizzano l'attributo '`th:`'.

Queste espressioni consentono di interpretare e manipolare i dati passati dal controller.

7.5 MongoDB

Il DBMS utilizzato a supporto dell'applicazione è **MongoDB**. MongoDB è un database non relazionale (NoSQL) che prevede una struttura “*schemaless*” e consente di memorizzare dati senza la necessità di definire uno schema rigido in anticipo, come avviene per database relazionali. Questa flessibilità è particolarmente vantaggiosa per lo sviluppo di applicazioni che possono evolvere rapidamente durante lo sviluppo, e quindi per progetti che prevedono metodologie agili, poiché riduce la necessità di apportare modifiche rilevanti alla struttura del database. Le richieste inviate mediante le interfacce DAO sono tradotte in operazioni CRUD (Create, Read, Update, Delete) sui dati all'interno del database.

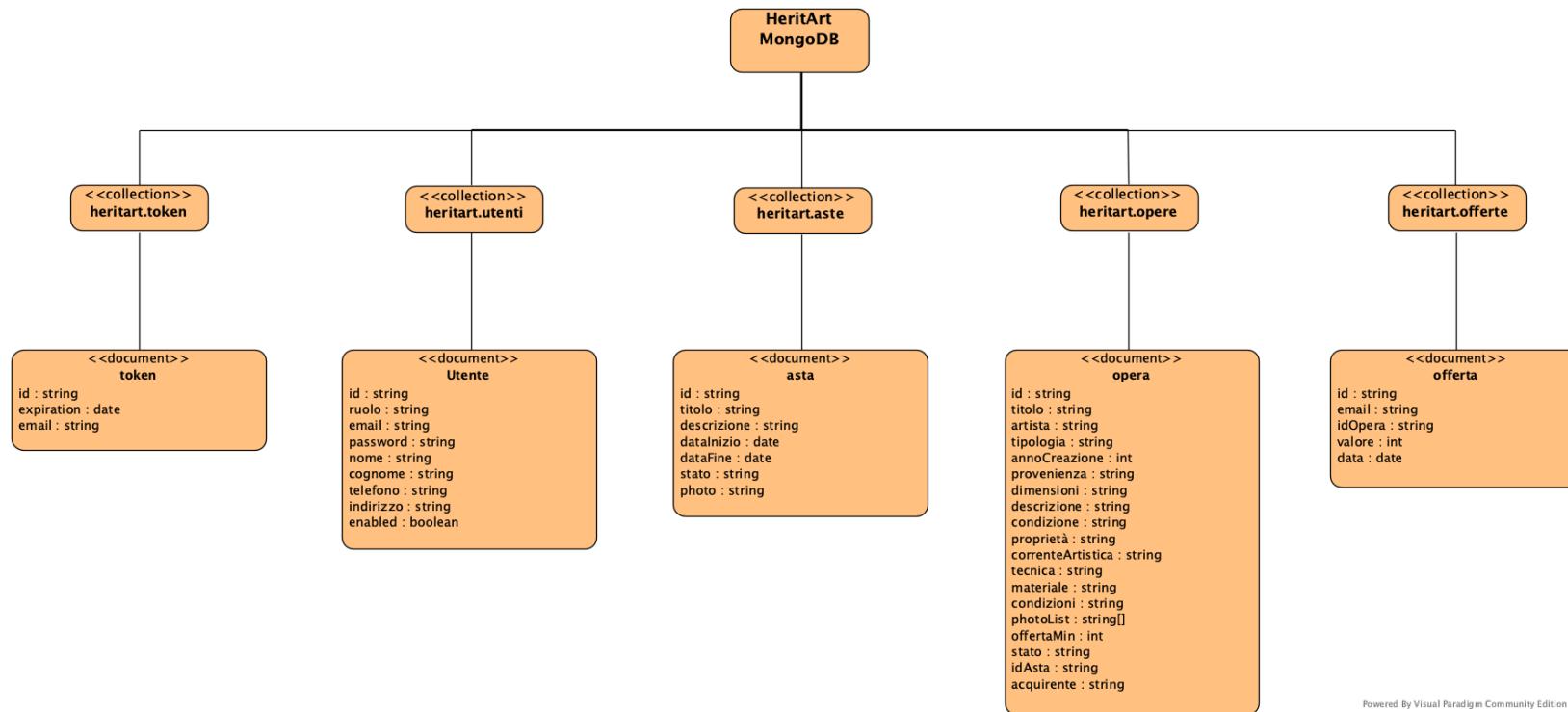
Di seguito sono riportati esempi di documenti BSON appartenenti alle collection “opere” e “aste”.

```
▼ {  
  ▼ "_id": {  
    "$oid": "64dd37b66ae17b75c8b91541"  
  },  
  "titolo": "Nascita di Venere",  
  "artista": "Sandro Botticelli",  
  "annoCreazione": 1485,  
  "provenienza": "Firenze",  
  "tipologia": "Dipinto",  
  "dimensioni": "175 x 278 (cm)",  
  "descrizione": "Opera iconica del Rinascimento italiano,  
  "proprietà": "COLLEZIONE_PUBBLICA",  
  "correnteArtistica": "RINASCIMENTO",  
  "tecnica": "OLIO",  
  "condizioni": "OTTIME",  
  "photoList": [ ],  
  "offerta": 1050,  
  "stato": "ASTA",  
  "idAsta": "64dd38bd6ae17b75c8b91544",  
  "_class": "com.heritart.model.opere.Opera"  
}  
  
▼ {  
  ▼ "_id": {  
    "$oid": "64dd38076ae17b75c8b91542"  
  },  
  "titolo": "Guernica",  
  "artista": "Pablo Picasso",  
  "annoCreazione": 1937,  
  "provenienza": "Madrid",  
  "tipologia": "Dipinto",  
  "dimensioni": "349 x 776 (cm)",  
  "descrizione": "L'ispirazione per l'opera, improvvisa e  
  "proprietà": "COLLEZIONE_PUBBLICA",  
  "correnteArtistica": "CUBISMO",  
  "tecnica": "OLIO",  
  "condizioni": "OTTIME",  
  "photoList": [ ],  
  "offerta": 850,  
  "stato": "DISPONIBILE",  
  "_class": "com.heritart.model.opere.Opera"  
}  
  
▼ {  
  ▼ "_id": {  
    "$oid": "64da94ac154fba0f65e64f83"  
  },  
  "titolo": "Asta 1",  
  "idGestore": "64da94ac154fba0f65e64f83",  
  "descrizione": "Asta imperdibile",  
  ▼ "dataInizio": {  
    "$date": "2023-08-23T06:00:00.000Z"  
  },  
  ▼ "dataFine": {  
    "$date": "2023-09-25T09:15:00.000Z"  
  },  
  "stato": "IN_CORSO",  
  "photo": "/9j/4AAQSkZJRgABAQAAAQABAAAD/2wCEAAoHCBcWFNgWFhY",  
  "_class": "com.heritart.model.aste.Asta"  
}  
  
▼ {  
  ▼ "_id": {  
    "$oid": "64de0df5388571187b003742"  
  },  
  "titolo": "Asta 2",  
  "idGestore": "64de0ccb388571187b00373f",  
  "descrizione": "Asta imperdibile",  
  ▼ "dataInizio": {  
    "$date": "2023-08-23T06:00:00.000Z"  
  },  
  ▼ "dataFine": {  
    "$date": "2023-08-27T10:00:00.000Z"  
  },  
  "stato": "TERMINATA",  
  "photo": "/9j/4AAQSkZJRgABAQEAYABgAAD/2wCEAAgGBgcGBQgHBwc",  
  "_class": "com.heritart.model.aste.Asta"  
}
```

8. Modello dei dati

Il modello dei dati fornisce una rappresentazione della struttura del database. I dati sono organizzati in collection, memorizzati come documenti BSON, simile al più noto formato JSON, e facilmente scambiabili con l'applicazione mediante le API di MongoDB.

Essendo un database “*schemaless*”, è possibile inserire nella stessa collection documenti con strutture eterogenee, che prevedono attributi tipizzati. Essendo un database NoSQL, le relazioni tra documenti non sono definite in modo esplicito, ma possono essere imposte grazie agli attributi. Ad esempio, è possibile inserire come attributo di un'opera l'id dell'asta di appartenenza o di un'acquirente.

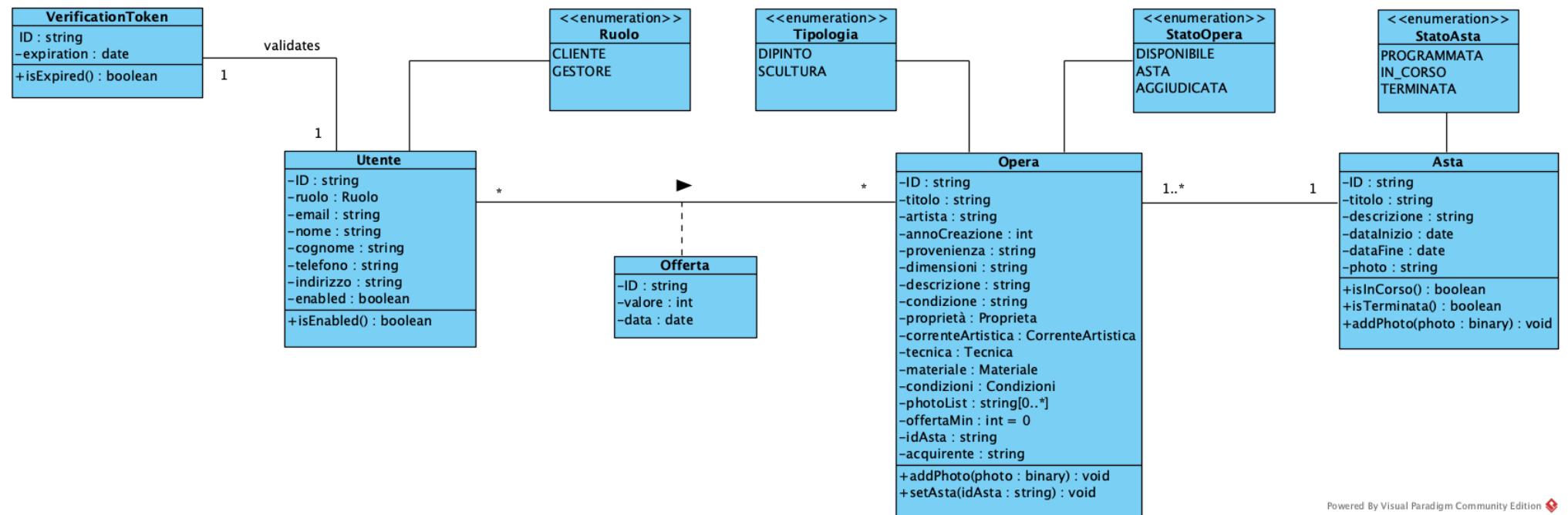


Powered By Visual Paradigm Community Edition

9. Modello di progetto

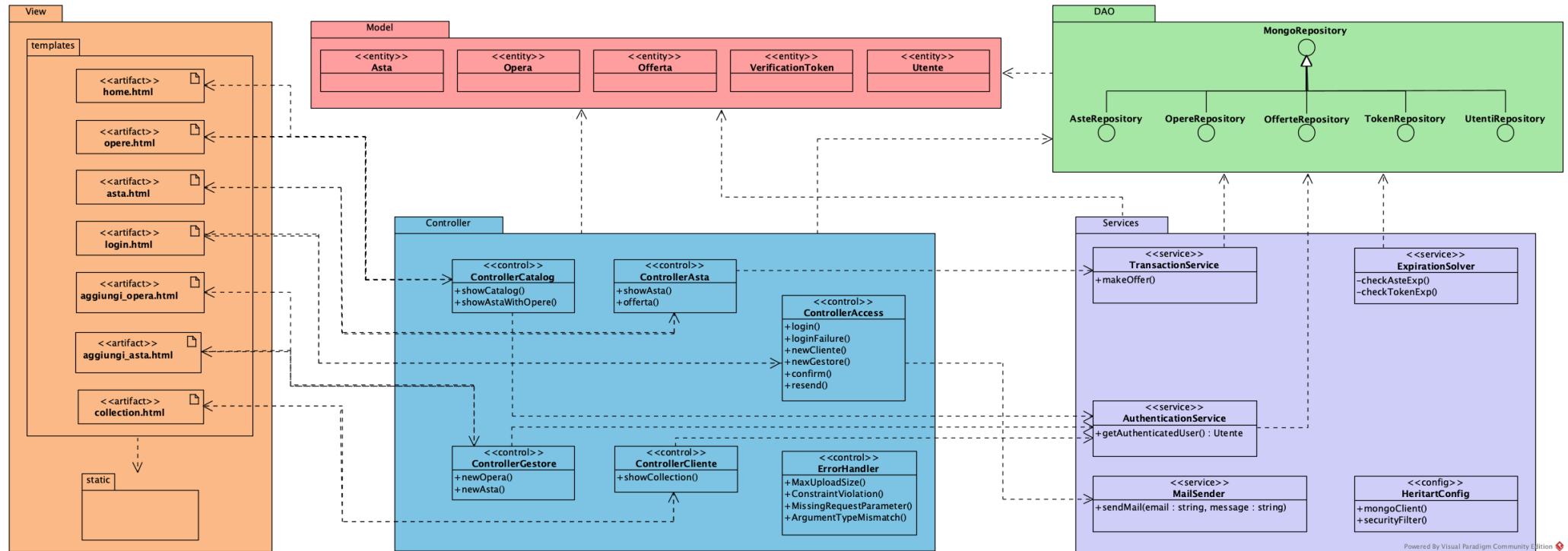
9.1 Class Diagram

Il Class Diagram fornisce una rappresentazione tecnica del dominio del progetto, mostrando le caratteristiche e le relazioni tra le classi previste dal Model.



9.2 Package Diagram

Il Package Diagram offre una panoramica delle principali classi implementate, mettendo in evidenza le loro dipendenze e la loro suddivisione, conforme allo schema architetturale MVC.



Model

Comprende tutte le classi rappresentative delle principali entità del dominio applicativo, descritte nel Class Diagram. Alle classe è associata la notazione *@Document*, che consente alle interfacce DAO di interpretare le istanze come documenti presenti nel database.

View

Raccoglie gli elementi di presentazione, ovvero le pagine HTML che realizzano l'interfaccia utente. Esse rendono possibili le interazioni degli utenti e l'invio delle richieste di accesso alle risorse web gestite dai controller. Ciascun controller, in base alla richiesta, stabilisce l'elemento della View da mostrare. Il supporto di Thymeleaf favorisce ulteriormente il coinvolgimento dei controller, con la possibilità di adattare dinamicamente i template inserendo dati del Model o del contesto applicativo.

DAO

Le interfacce DAO estendono l'interfaccia *MongoRepository* delle librerie Spring Data MongoDB, che include i metodi che realizzano le query verso il database MongoDB. È possibile effettuare l'override dei metodi ereditati o dichiararne altri automaticamente interpretati come query mediante l'utilizzo di parole chiave (ad esempio, *findByTitoloAndStato()*). Le query di ricerca ricevono dati dal database in formato BSON e li traducono in istanze del Model, o viceversa per le query di inserimento o modifica. Sono previste tante interfacce DAO quante sono le collezioni del database.

Control

Includono tutte le classi indicate come *@Controller*, delegate alla gestione delle richieste di accesso alle risorse web. Una risorsa è definita da un percorso specifico (endpoint REST) e da un metodo http (GET o POST). Ad ogni risorsa esposta da un controller è associato un metodo che esegue tutte le operazioni necessarie per servire una richiesta. Le risorse accessibili sono distribuite tra diversi controller, ognuno dei quali assiste una specifica sezione dell'applicazione. Tale suddivisione facilita l'organizzazione del codice e l'assegnazione dei compiti tra i componenti del team.

I controller fanno ampiamente utilizzo delle repository DAO e dei servizi.

- **ControllerAccess:** si occupa dell'accesso e della registrazione degli utenti. Gestisce l'attivazione degli account mediante la generazione di token e l'invio di mail di conferma.
- **ControllerCatalog:** mostra agli utenti il catalogo delle aste e gestisce la ricerca, mostra la lista di opere di un'asta selezionata.
- **ControllerCliente:** mostra la collezione personale di opere aggiudicate da un cliente.
- **ControllerGestore;** consente ad un gestore di aggiungere un'opera o un'asta al catalogo.
- **ControllerAsta:** mostra i dettagli di un'opera all'asta con l'elenco delle ultime migliori offerte, gestisce l'invio delle offerte da parte dei clienti durante un'asta in corso.
- **ErrorHandler:** gestisce gli errori che possono verificarsi durante l'elaborazione delle richieste, restituendo risposte di errore appropriate con codici di stato HTTP corrispondenti. Ciò semplifica la validazione dei dati in input e lo svolgimento dei test. Le eccezioni gestite riguardano, in particolare, i parametri inviati in seguito alla compilazione di una form. Un parametro previsto per una determinata richiesta servita da un controller può essere connotato con un'espressione che ne indica un *constraint*, ad esempio `@NotBlank` per un parametro testuale. In questo modo, la violazione del vincolo può essere rilevata dall'ErrorHandler. In aggiunta, esso corregge automaticamente il formato delle stringhe e delle date fornite. L'ErrorHandler intercetta violazioni dovute a:
 - *MissingRequestParameter*: assenza di un parametro obbligatorio;
 - *ConstraintViolation*: vincolo indicato per un parametro non rispettato;
 - *ArgumentTypeMismatch*: parametro non valido;
 - *MaxUploadSize*: superamento della massima dimensione consentita per i file caricati.

In seguito ad una violazione, l'ErrorHandler restituisce all'utente un opportuno messaggio di errore.

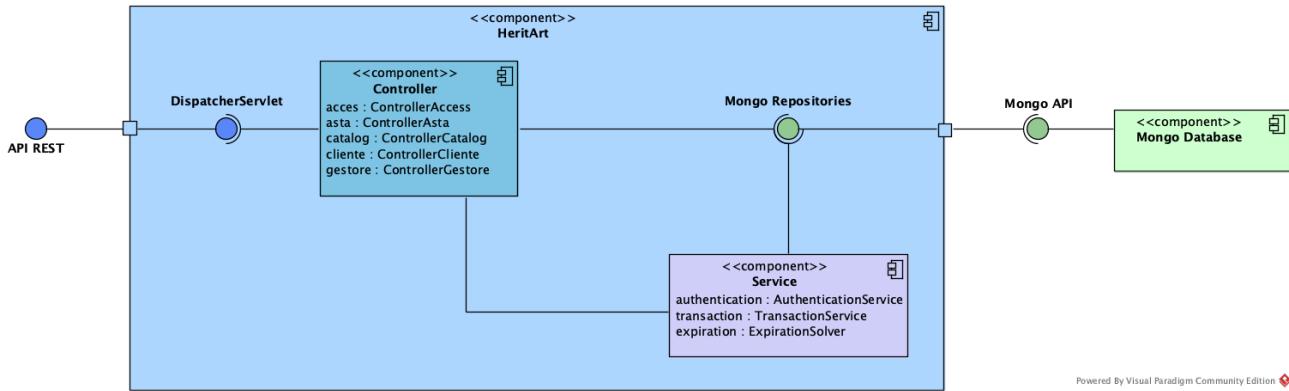
Services

Contiene tutte le classi, dichiarate con notazioni speciali come `@Service` o `@Component`, che forniscono uno specifico contributo alla logica operativa gestita dai controller.

- **HeritartConfig:** rappresenta la classe di configurazione dell'applicazione, identificata dalla notazione `@Configuration`. Essa consente di modificare alcune proprietà dell'applicazione alternative a quelle automaticamente imposte dal framework Spring Boot.
Essa stabilisce la connessione verso il database. Inoltre, sfruttando le librerie di sicurezza di Spring e il servizio AuthenticationService, sono gestite la modalità di autenticazione e le autorizzazioni per l'accesso alle risorse dell'applicazione.
In particolare, sono indicati quali percorsi sono riservati al cliente, al gestore o a entrambi.
- **AuthenticationService:** si occupa del processo di validazione di un utente che effettua il login, verificando che l'account sia correttamente registrato e attivato. È possibile richiamare tale servizio per ottenere il ruolo o altre informazioni relative ad un utente loggato.
- **MailSender:** è la classe che consente di inviare mail agli utenti, mediante il server SMTP di Google. Ciò avviene per la conferma di una registrazione o per comunicare al miglior offerente il conferimento di un'opera al termine di un'asta.
- **TransactionService:** risolve la possibile presentazione di offerte concomitanti per un'opera all'asta. La notazione `@Transactional` applicata alla classe indica che i metodi rappresentano uniche transazioni con le relative operazioni verso il database. Una nuova offerta viene accettata se è superiore all'ultima migliore offerta presentata.
- **ExpirationSolver:** gestisce autonomamente le scadenze previste dall'applicazione, cioè quelle inerenti alle aste e ai token. La notazione `@Scheduled` consente di specificare la periodicità con cui tale servizio interviene controllando, in base alle date, quali aste iniziare o terminare e quali token sono scaduti. Per ciascuna occorrenza sono previste opportune operazioni come la variazione dello stato delle aste o delle opere.
La frequenza imposta per i controlli è di un minuto.

9.3 Component Diagram

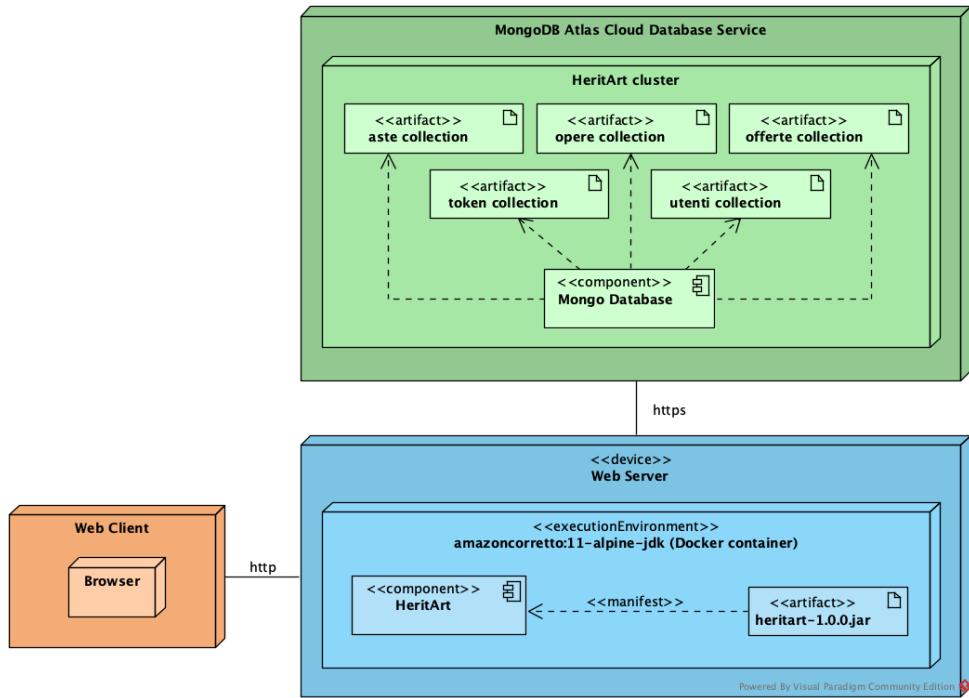
Il Component Diagram serve a visualizzare la struttura dell'applicazione in termini di componenti software, mettendo in risalto i meccanismi di comunicazione sia interni che esterni.



I componenti interni sono quelli caratteristici dell'architettura MVC adottata e rappresentano costrutti logici esistenti in virtù del fatto che le loro parti sono istanziate. L'interazione tra i client e l'applicazione avviene mediante API REST, che prevedono l'utilizzo di metodi HTTP e lo scambio di dati in formato HTML o JSON. Le richieste sono intercettate dal *DispatcherServlet*, un modulo automaticamente configurato da Spring Boot. Esso sostituisce il descrittore di deployment (web.xml) tipico delle architetture REST tradizionali, in cui le associazioni tra servlet, risorse e pagine web sono definite in maniera esplicita. Il *DispatcherServlet* converte i parametri di una richiesta in tipi Java, invoca il controller e l'endpoint REST in grado di soddisfare la richiesta e fornisce gli elementi della view da mostrare all'utente. L'invio di query al database avviene mediante le interfacce DAO (Repositories). In particolare, le query di ricerca ricevono dati dal database in formato BSON e li traducono in istanze del Model, o viceversa per le query di inserimento o modifica. Attraverso le API messe a disposizione da MongoDB, le richieste provenienti dall'applicazione sono tradotte in operazioni CRUD (Create, Read, Update, Delete) sui dati del database.

9.4 Deployment Diagram

Il diagramma di deployment mostra l'ambiente di distribuzione previsto per il rilascio su rete del sistema.

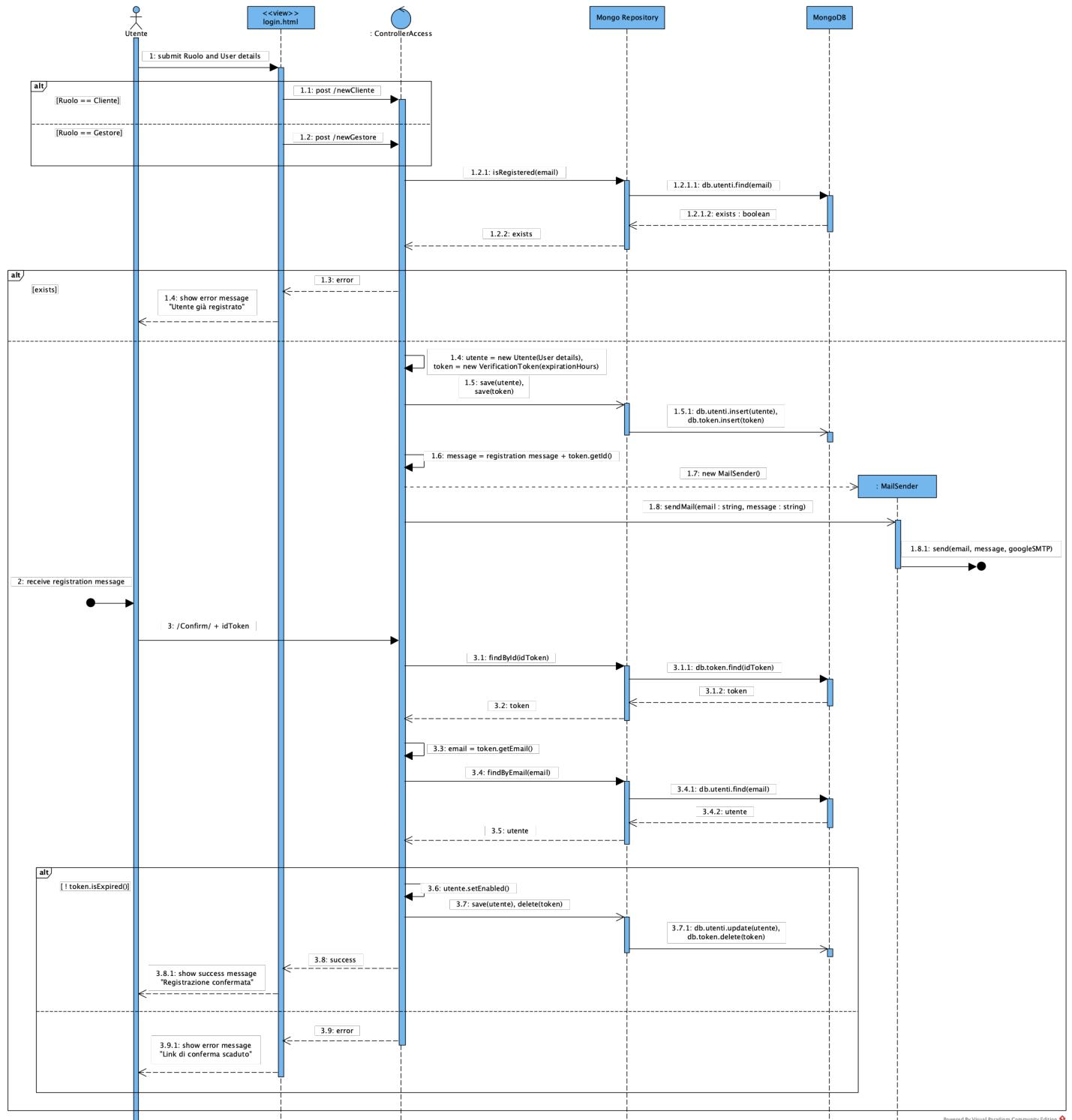


L'applicazione HeritArt può essere compressa ed eseguita come file JAR. La piattaforma di virtualizzazione Docker può essere utilizzata per realizzare e pubblicare un container che includa tale file, con ambiente operativo Alpine Linux e OpenJDK 11 (library e compilatore Java).

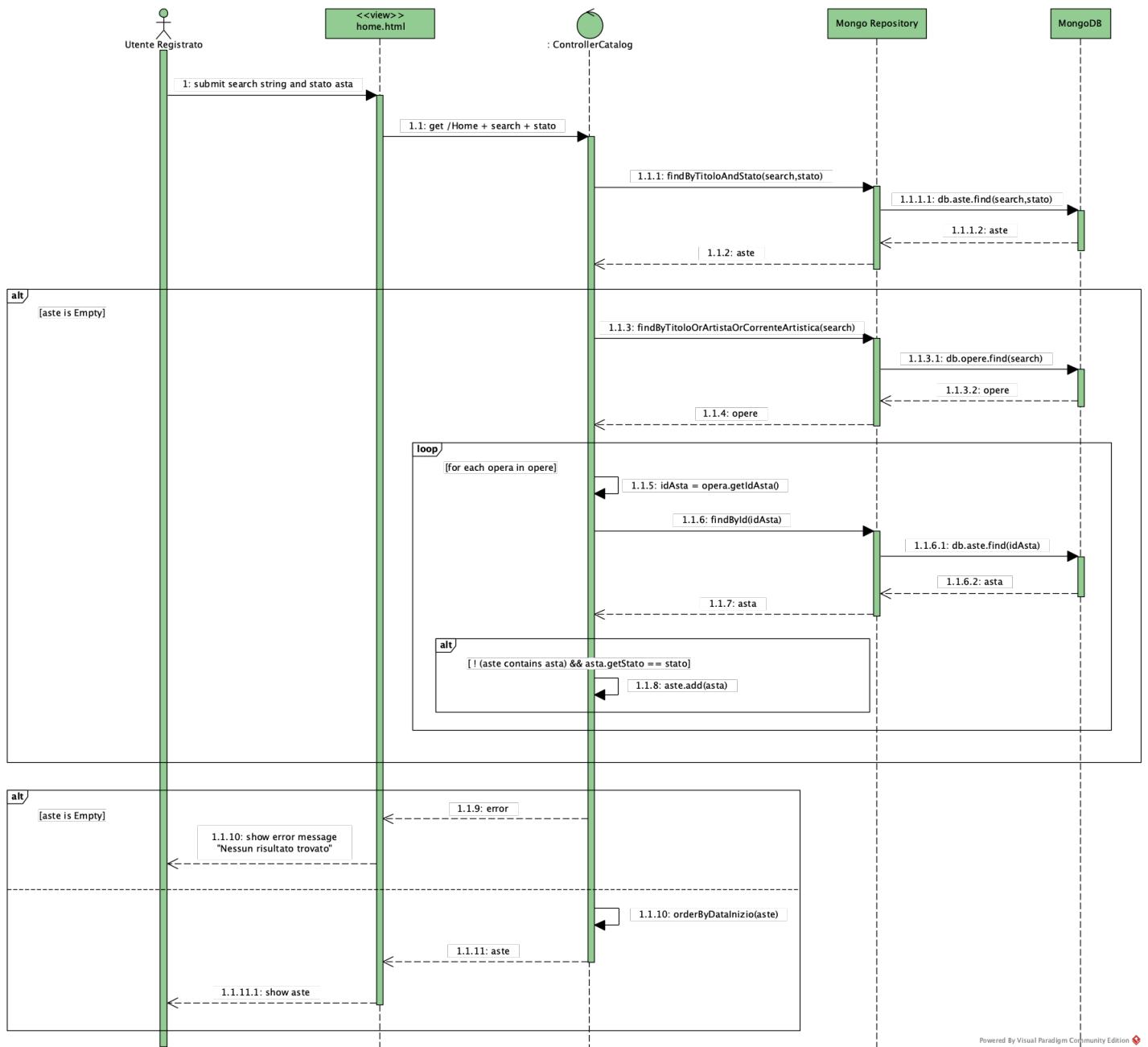
Il container può essere caricato e messo in esecuzione su un qualsiasi web server. Nel nostro caso, è stato utilizzato un server fornito dalla piattaforma cloud Microsoft Azure. Un utente può accedere all'applicazione collegandosi al server tramite un generico web browser. Il database Mongo e le collection necessarie sono distribuite su un cluster istanziato nell'infrastruttura cloud Mongo Atlas. Tale servizio consente di monitorare, gestire e scalare comodamente il DBMS in base alle esigenze riscontrabili in seguito al rilascio.

9.5 Sequence Diagram

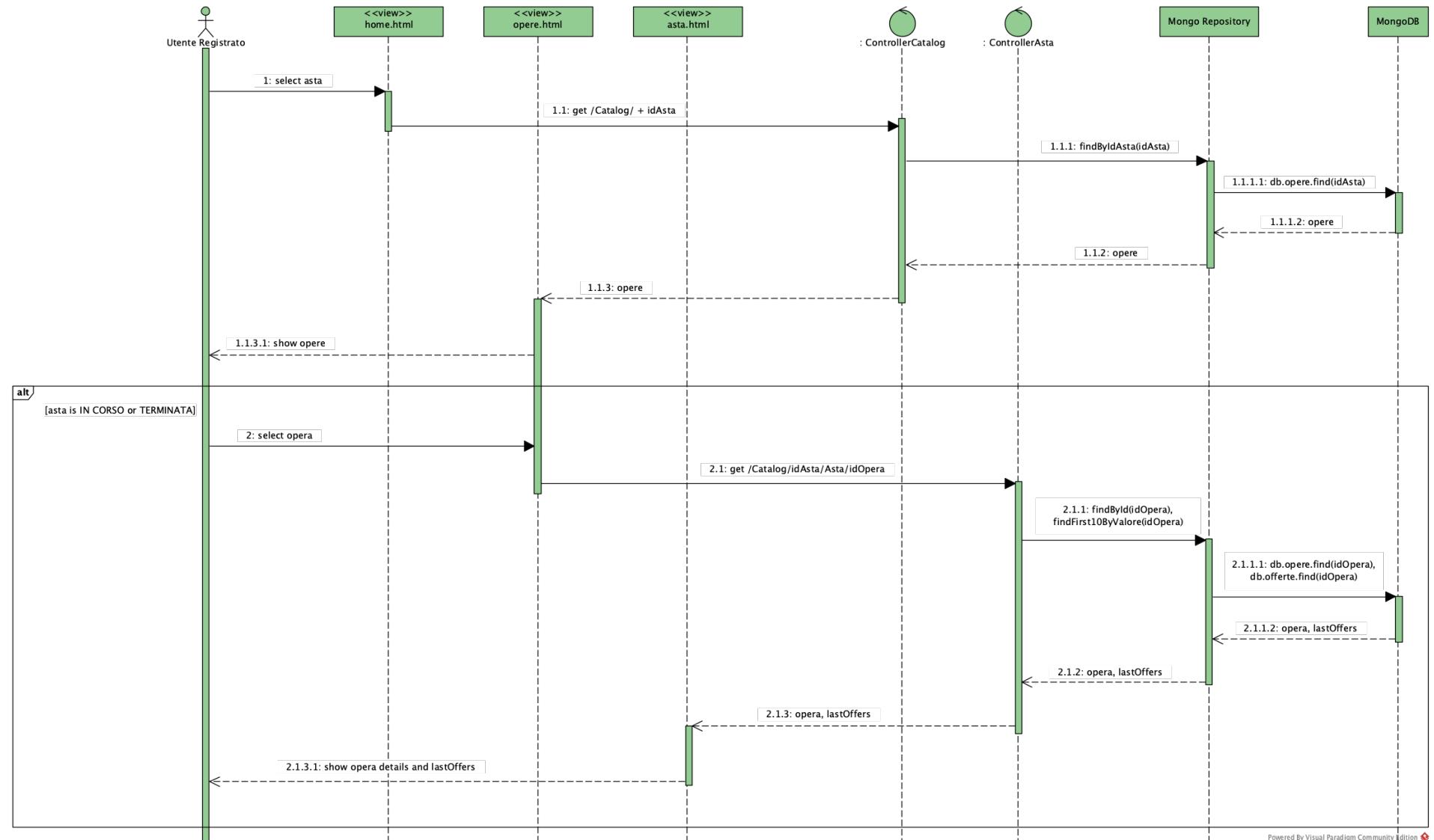
Registrazione



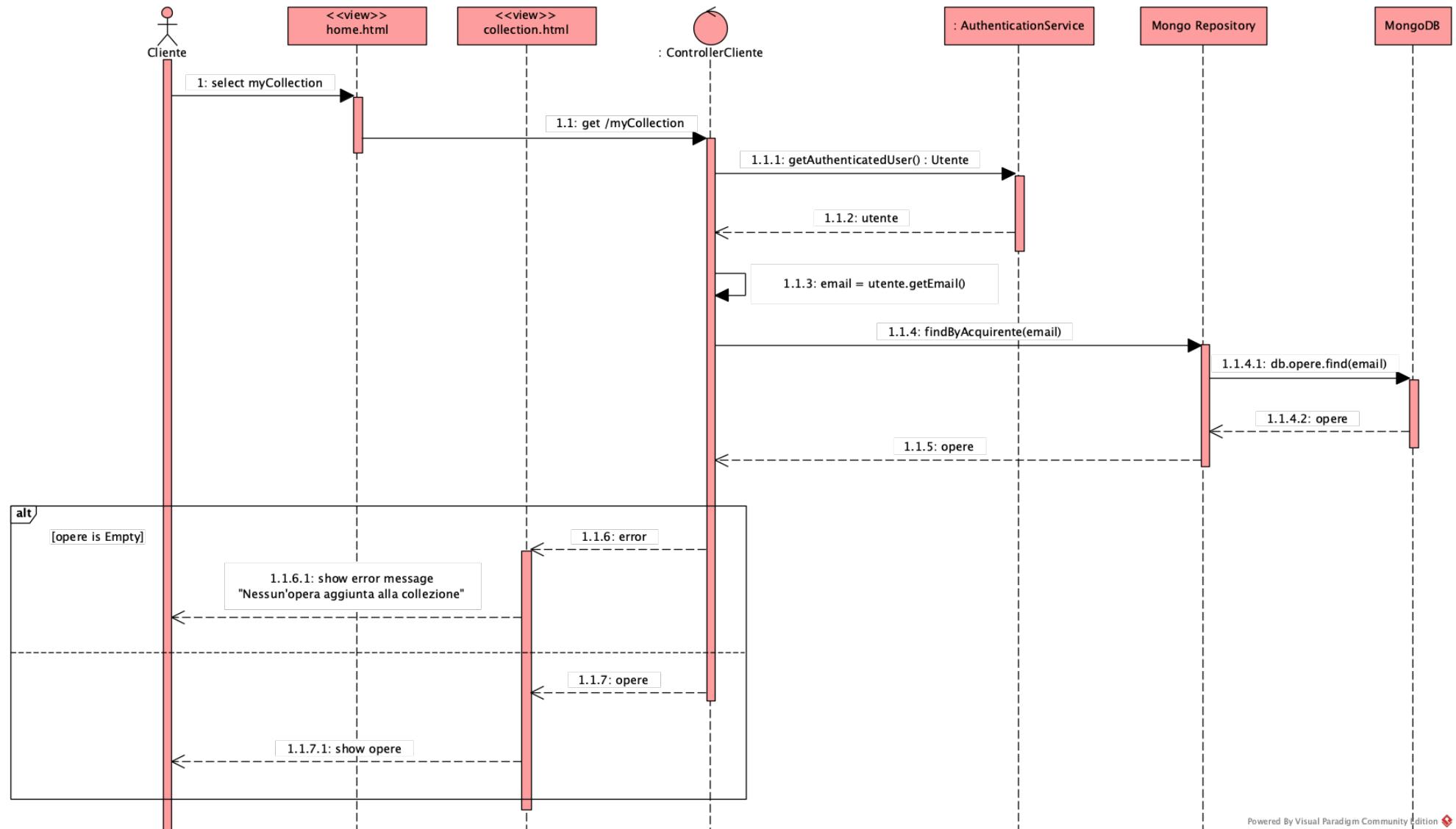
Ricerca



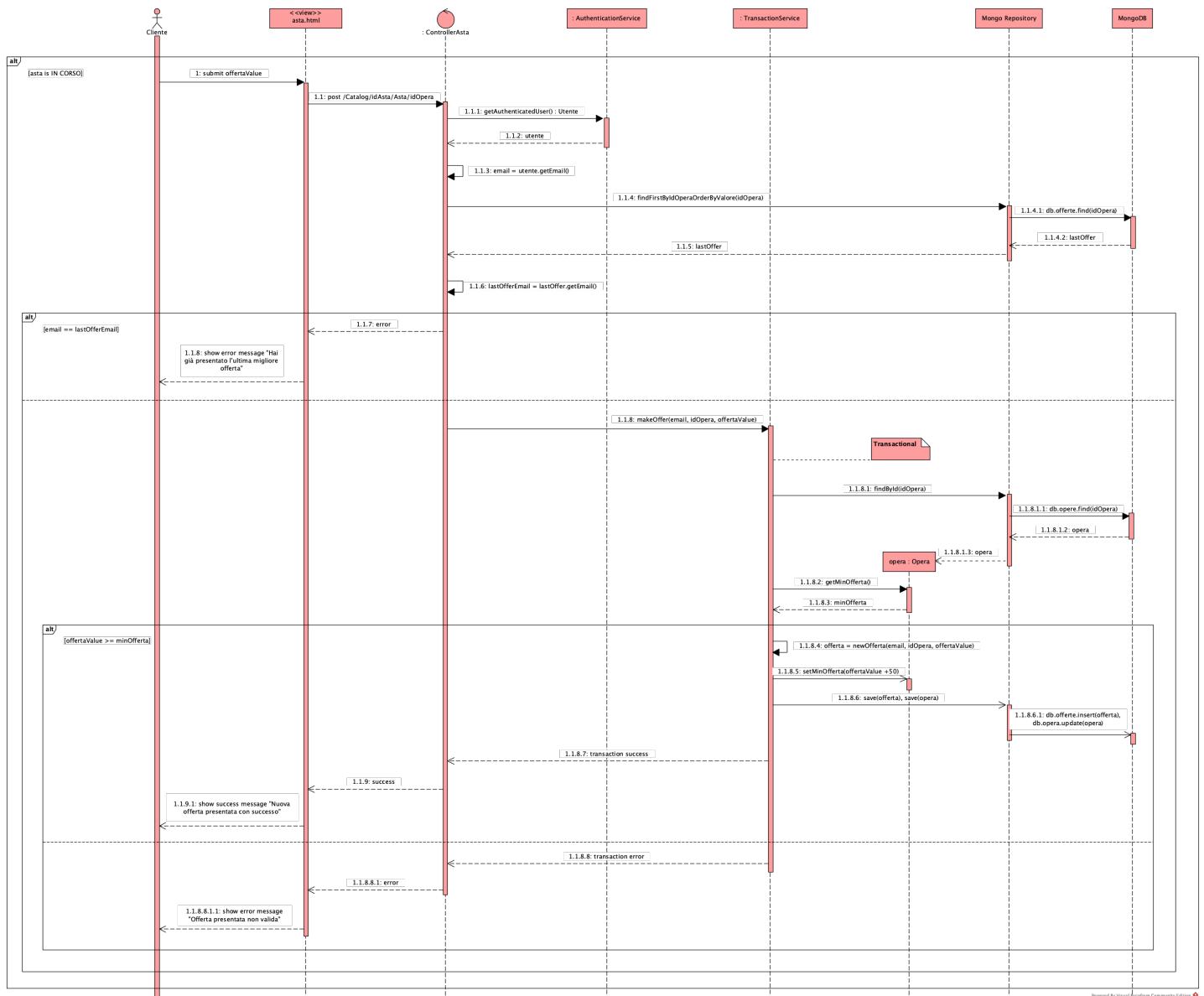
Visualizza Asta



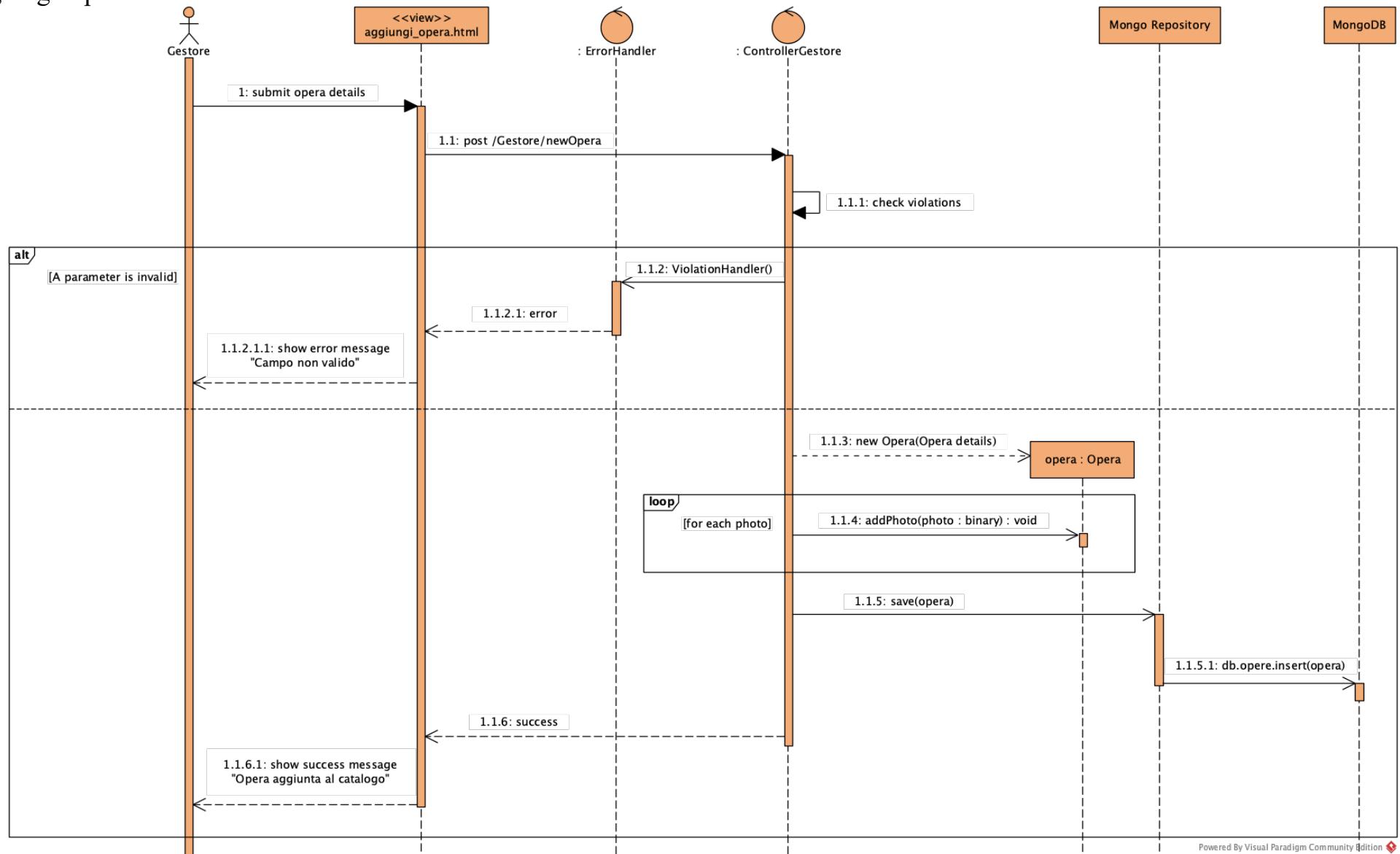
Visualizza Collezione



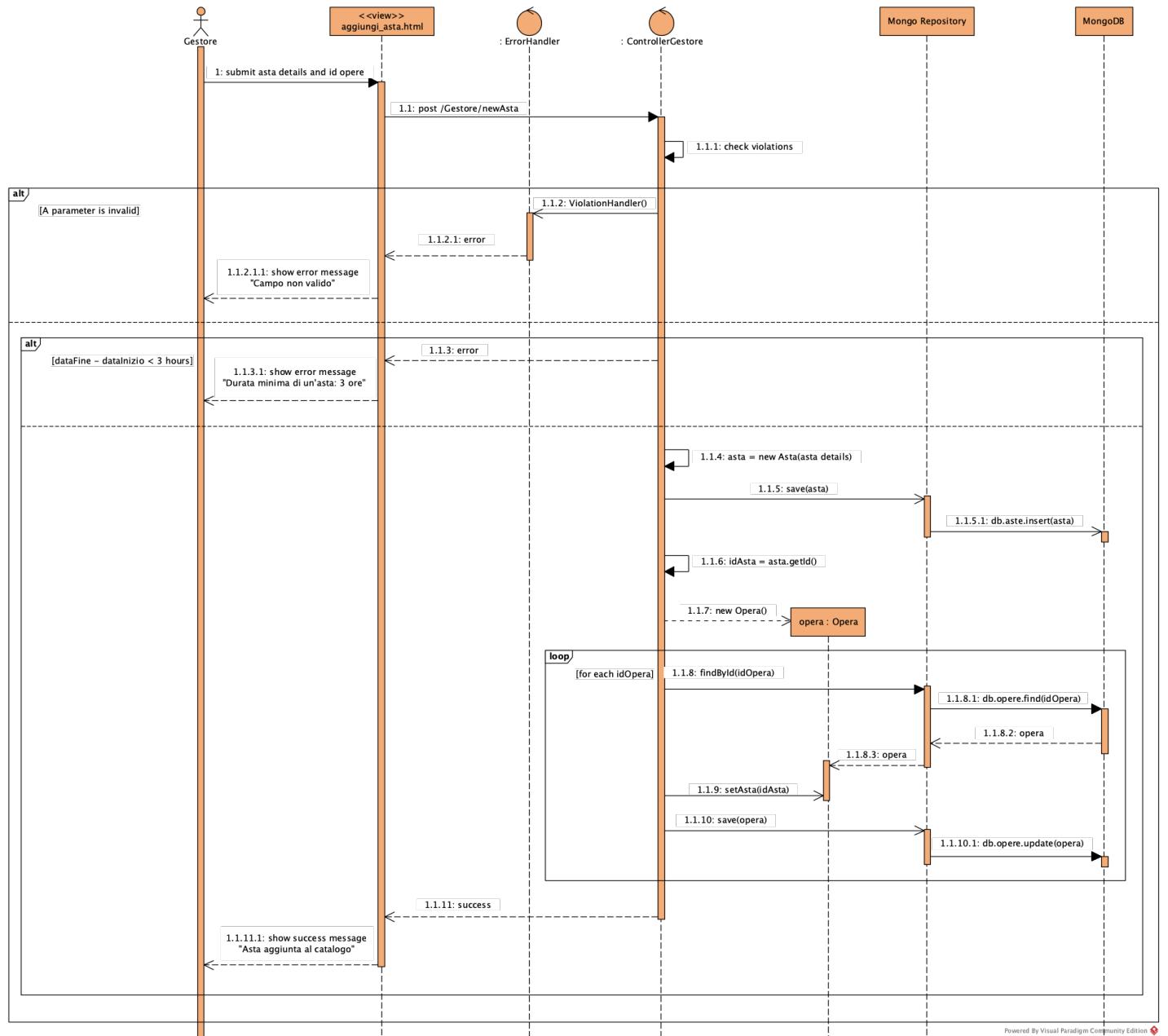
Offerta



Aggiungi Opera

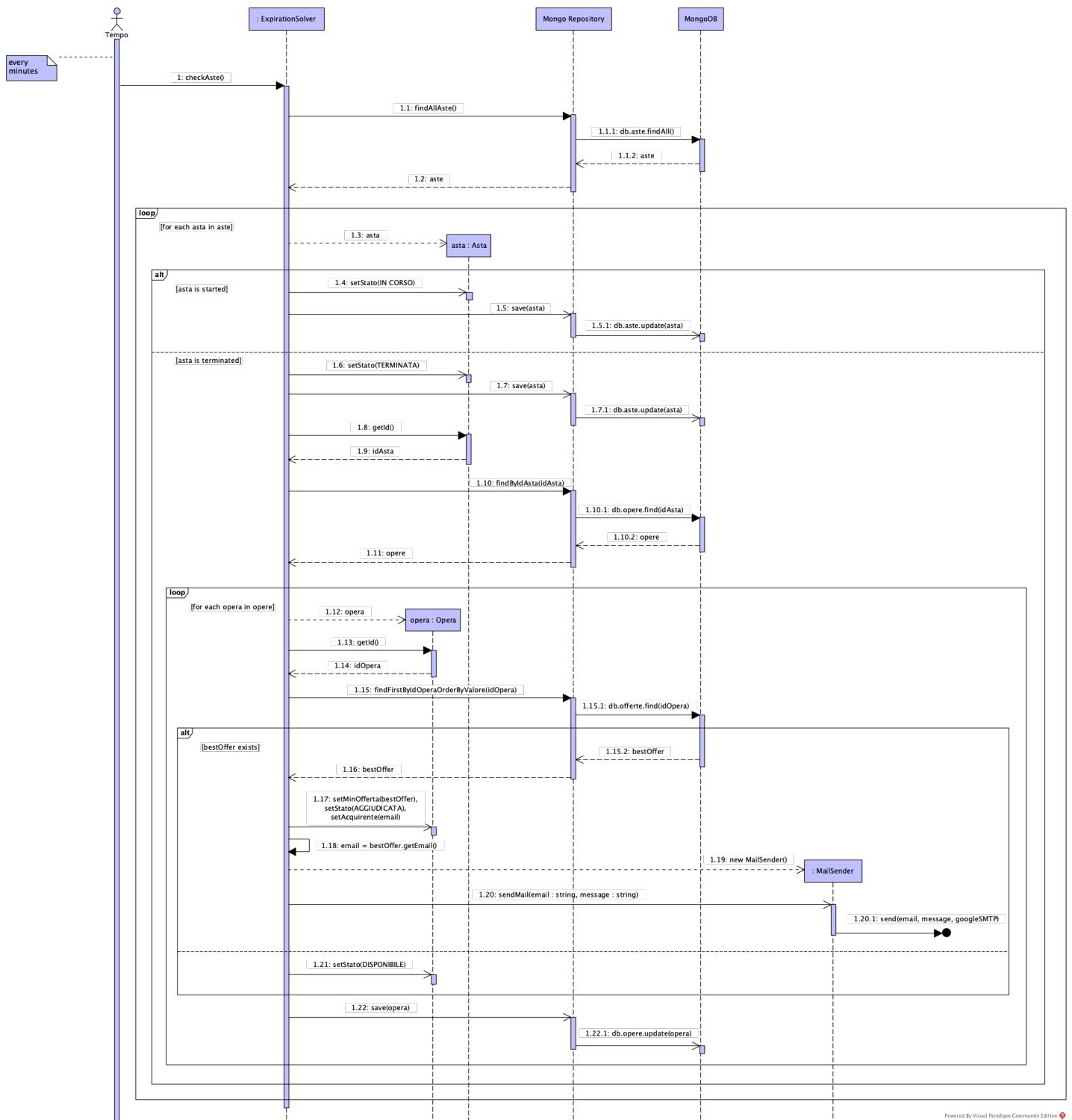


Aggiungi Asta



Powered By Visual Paradigm Community Edition

Gestisci aste



Powered By Visual Paradigm Community Edition

10. Testing funzionale

I test funzionali sono stati realizzati mediante:

- Il framework **JUnit**, che consente di testare singole unità di codice;
- Le librerie di supporto ai test di **Spring Boot**;
- **MockMVC**, che consente di testare i controller simulando richieste HTTP reali.

Sono stati testati i metodi che richiedono la validazione dei parametri forniti da una richiesta. I controller interessati sono:

- ControllerAccess per la registrazione di un utente;
- ControllerGestore per l'aggiunta di un'opera o un'asta da parte di un gestore;
- ControllerAsta per l'accettazione di un'offerta inviata da un cliente.

Le violazioni sono intercettate dall'ErrorHandler che fornisce i messaggi di errore attesi.

ControllerAccess

(POST /newCliente)

Test Case	Descrizione	Output atteso	HTTP status	Esito
CA01	Durante la registrazione, l'utente ha inserito un parametro <param> non valido.	"Campo non valido: <param>."	400: Bad Request	SUCCESS
CA02	Durante la registrazione, l'utente non ha inserito un parametro <param> obbligatorio.	"Inserire: <param>."	400: Bad Request	SUCCESS
CA03	Durante la registrazione, l'utente ha inserito correttamente tutti i parametri richiesti.	"Conferma la registrazione accedendo al link inviato a <email>."	302: Found redirect	SUCCESS
CA04	Durante la registrazione, l'utente ha inserito una email associata ad un account già registrato.	"L'utente <email> è già registrato."	302: Found redirect	SUCCESS

Test Case	Descrizione	Output atteso	HTTP status	Esito
CG01	Durante l'inserimento di un'opera, il gestore ha inserito correttamente tutti i parametri richiesti.	"Opera aggiunta al catalogo."	302: Found redirect	SUCCESS
CG02	Durante la l'inserimento di un'opera, il gestore ha inserito un parametro <param> non valido.	"Campo non valido: <param>."	400: Bad Request	SUCCESS
CG03	Durante la registrazione, l'utente non ha inserito un parametro <param> obbligatorio.	"Inserire: <param>."	400: Bad Request	SUCCESS
CG04	Durante l'inserimento di un'asta, il gestore ha inserito correttamente tutti i parametri richiesti.	"Asta aggiunta al catalogo. L'asta avrà inizio in data <dataInizio> alle ore <oraInizio> e terminerà in data <dataFine> alle ore <oraFine>."	302: Found redirect	SUCCESS
CG05	Durante la l'inserimento di un'asta, il gestore ha inserito un parametro <param> non valido.	"Campo non valido: <param>."	400: Bad Request	SUCCESS
CG06	Durante la l'inserimento di un'asta, il gestore non ha inserito un parametro <param> obbligatorio.	"Inserire: <param>."	400: Bad Request	SUCCESS
CG07	Durante la l'inserimento di un'asta, il gestore ha inserito <dataFine> a meno di 3 ore dopo <dataInizio>.	"Durata minima di un'asta: 3 ore. Fissare un termine ad almeno 3 ore dopo la data d'inizio."	302: Found redirect	SUCCESS

ControllerAsta

(POST /Catalog/{idAsta}/Asta/{idOpera}/)

Test Case	Descrizione	Output atteso	HTTP status	Esito
C001	Durante l'invio di un'offerta, il cliente ha presentato un valore minore dell'ultima migliore offerta.	"Offerta presentata non valida. Rinvia una nuova offerta."	302: Found redirect	SUCCESS
C002	Durante l'invio di un'offerta, il cliente ha presentato un'offerta valida.	"Nuova offerta presentata con successo."	302: Found redirect	SUCCESS
C003	Durante l'invio di un'offerta, il cliente ha già presentato l'ultima migliore offerta.	"Hai già presentato l'ultima migliore offerta. Attendi una nuova offerta."	302: Found redirect	SUCCESS

```
com.heritart.control in heritart: 14 total, 14 passed
48.63 s
Collapse | Expand

ControllerAccessTest
3.71 s
newCliente1() passed 512 ms
newCliente2() passed 9 ms
newCliente3() passed 2.83 s
newCliente4() passed 364 ms

ControllerGestoreTest
12.47 s
newOpera1() passed 287 ms
newOpera2() passed 10 ms
newOpera3() passed 9 ms
newAsta1() passed 11.59 s
newAsta2() passed 72 ms
newAsta3() passed 7 ms
newAsta4() passed 499 ms

ControllerAstaTest
32.44 s
Offerta1() passed 6.85 s
Offerta2() passed 13.58 s
Offerta3() passed 12.01 s

Generated by IntelliJ IDEA on 06/09/23, 15:25
```

11. Istruzioni per l'installazione

11.1 Creazione dell'immagine Docker

Requisiti: Maven e Docker Engine (o Docker Desktop)

1. Aprire il terminale nella cartella di progetto “heritart”;
2. Per rimuovere i file generati nelle build precedenti, ricompilare il codice sorgente e creare il file JAR dell'applicazione, eseguire il comando `mvn clean package` ;
3. Per verificare il funzionamento dell'applicazione è possibile runnare il file JAR con il comando `sudo java -jar target/heritart-0.0.1-SNAPSHOT.jar` ;
4. La cartella di progetto contiene il Dockerfile, un documento con le specifiche per la creazione del container, tra cui l'ambiente operativo (Alpine Linux con OpenJDK 11) e il riferimento al file JAR;
5. Il comando `docker build --tag=cruelsuerte/heritart:latest .` consente di creare l'immagine Docker a partire dal Dockerfile, associando ad essa un tag. Il tag fa riferimento ad una repository pubblica di Docker Hub;
6. Per caricare l'immagine su Docker Hub eseguire il comando
`docker push cruelsuerte/heritart .`

11.2 Installazione del container in locale

Requisiti: Docker Engine (o Docker Desktop)

7. Per scaricare l'immagine dalla repository di Docker Hub, aprire il terminale ed eseguire il comando `docker pull cruelsuerte/heritart` ;
8. Avviare il container eseguendo il comando
`docker run -p80:80 cruelsuerte/heritart .`

11.3 Deploy su Microsoft Azure

Requisiti: sottoscrizione Microsoft Azure

1. Autenticarsi al portale Microsoft Azure;
2. Accedere a “Servizi app”;
3. Selezionare “Crea > App web”;
4. Compilare i campi relativi alle informazioni di base, indicando la sottoscrizione da utilizzare, il nome del gruppo di risorse, il nome dell’applicazione (‘heritart’), il tipo di istanza (contenitore docker) e la risorsa di calcolo desiderata;
5. Nella sezione Docker, indicare come origine Docker Hub, specificare il riferimento all’immagine Docker (‘cruelsuerte/heritart’) e abilitare l’accesso pubblico;
6. Creare l’istanza e attendere il completamento della distribuzione;
7. L’applicazione è accessibile al link <https://heritart.azurewebsites.net> .