QB SOFTWARE

×

UNIVERSITÀ DEGLI STUDI DI PADOVA

CORSO DI INGEGNERIA DEL SOFTWARE

ANNO ACCADEMICO 2023/2024

Specifica Tecnica

Contatti: qbsoftware.swe@gmail.com

Registro delle modifiche

V.	Data	Membro	Ruolo	Descrizione
1.0.0	05/05/2024	A. Domuta	Responsabile	Approvazione documento
0.11.0	27/04/2024	A. Feltrin	Verificatore	Controllo qualità
	26/04/2024	A. Bustreo	Progettista	Effettuato la revisione
0.10.0	25/04/2024	A. Feltrin	Verificatore	Controllo qualità
	25/04/2024	A. Giurisato	Progettista	Redatto la sezione API
0.9.0	24/04/2024	A. Domuta	Verificatore	Controllo qualità
	24/04/2024	S. Destro	Progettista	Aggiornato Specifica Tecnica
0.8.0	13/04/2024	A. Domuta	Verificatore	Controllo qualità
	12/04/2024	A. Bustreo	Progettista	Aggiornato la sezione dell'architettura
0.7.0	11/04/2024	A. Giurisato	Verificatore	Controllo qualità
	11/04/2024	A. Domuta	Progettista	Aggiunto i diagrammi delle classi
0.6.0	06/04/2024	A. Domuta	Verificatore	Controllo qualità
	05/04/2024	R. Fontana	Progettista	Espanso la sezione relativa all'architettura
0.5.0	05/04/2024	S. Destro	Verificatore	Controllo qualità
	04/04/2024	A. Giurisato	Progettista	Redatto la sezione relativa al database
0.4.0	23/03/2024	S. Rovea	Verificatore	Controllo qualità

V.	Data	Membro	Ruolo	Descrizione
	22/03/2024	R. Fontana	Progettista	Stesura sezione sui design pattern
0.3.0	21/03/2024	A. Domuta	Verificatore	Controllo qualità
	21/03/2024	S. Rovea	Progettista	Prima stesura sezione architettura
0.2.0	17/03/2024	A. Feltrin	Verificatore	Controllo qualità
	16/03/2024	A. Giurisato	Progettista	Redatto la sezione delle tecnologie
0.1.0	16/03/2024	A. Feltrin	Verificatore	Controllo qualità
	15/03/2024	S. Destro	Progettista	Redatto la sezione introduzione

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo e struttura

Con questo documento QB Software intende elencare e motivare le tecnologie, le scelte architetturali e i design pattern$_G$ scelti per lo sviluppo del progetto commissionato. In seguito verranno descritte le tecnologie impiegate, una lista degli endpoint accessibili e come interagire con essi.

Prima di leggere un qualunque documento prodotto da QB Software è necessario conoscere il significato dei termini riportati nel documento: *Glossario v2.0.0* presente nella repository: Documentazione [Online - GitHub; ultima visita 15/03/2024].

1.2 Riferimenti

1.2.1 Normativi

- *Norme di Progetto v2.0.0*;
- Standard ISO/IEC 12207:1997
  - https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
    [Online - PDF; ultima visita 15/02/2024];
- Capitolato d'appalto C8:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C8p.pdf
    [Online - PDF; ultima visita 30/11/2023];
- Regolamento del progetto didattico:
  - https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
    [Online - PDF; ultima visita 15/03/2024].

1.2.2 Informativi

- *Glossario v2.0.0*;

- *Analisi dei Requisiti v2.0.0*;

- Dispense dell'insegnamento d'Ingegneria del Software e approfondimenti:

  - Progettazione, le dipendenze fra le componenti:
    https://www.math.unipd.it/~rcardin/swea/2023/Object-Oriented%20Progamming%20Principles%20Revised.pdf
    [Online - PDF; ultima visita 15/03/2024];

  - Dependency Management in Object-Oriented Programming:
    https://www.math.unipd.it/~rcardin/swea/2022/Dependency%20Management%20in%20Object-Oriented%20Programming.pdf
    [Online - PDF; ultima visita 15/03/2024];

  - Progettazione e programmazione, diagrammi delle classi (UML_G):
    https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf
    [Online - PDF; ultima visita 15/03/2024];

  - Dependency:
    http://blog.rcard.in/programming/oop/software-engineering/2017/04/10/dependency-dot.html
    [Online - Blog; ultima visita 22/03/2024];

  - Progettazione, i pattern architetturali:
    https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf
    [Online - PDF; ultima visita 15/03/2024];

  - Hexagonal Archtecture Example:
    https://github.com/rcardin/hexagonal
    [Online - Repository; ultima visita 15/03/2024];

  - Progettazione, il pattern Dependency Injection:
    https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Architetturali%20-%20Dependency%20Injection.pdf

[Online - PDF; ultima visita 22/03/2024];

- Lezione T6 - Progettazione software:
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf
  [Online - PDF; ultima visita 15/03/2024];

- Fan-in e fan-out:
  https://www.math.unipd.it/~tullio/IS-1/2004/Approfondimenti/Fan-in_Fan-out.html
  [Online - Blog; ultima visita 15/03/2024];

- Definizioni di "architettura software":
  https://www.math.unipd.it/~tullio/IS-1/2006/Approfondimenti/SEI-Software_Architectures.pdf
  [Online - PDF; ultima visita 15/03/2024];

- Christopher Alexander sulla teoria dei pattern:
  https://www.math.unipd.it/~tullio/IS-1//2008/Approfondimenti/Christopher_Alexander_1999.pdf
  [Online - PDF; ultima visita 22/03/2024];

- Progettazione, i pattern creazionali (GoF):
  https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf
  [Online - PDF; ultima visita 22/03/2024];

- Progettazione, i pattern strutturali (GoF):
  https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf
  [Online - PDF; ultima visita 22/03/2024];

- Progettazione, i pattern di comportamento (GoF):
  https://www.math.unipd.it/~rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf
  [Online - PDF; ultima visita 22/03/2024];

  - Programmazione, SOLID programming:

    https://www.math.unipd.it/~rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf

    [Online - PDF; ultima visita 15/03/2024];

  - Single-Responsibility Principle done right:

    http://blog.rcard.in/solid/srp/programming/2017/12/31/srp-done-right.html

    [Online - Blog; ultima visita 22/03/2024];

- Approfondimenti online:

  - Specifiche JMAP_{G}:

    https://jmap.io/spec.html

    [Online - Docs; ultima visita 16/03/2024];

  - Manuale MongoDB:

    https://www.mongodb.com/docs/manual/

    [Online - Docs; ultima visita 16/03/2024];

  - Manuale Jetty:

    https://eclipse.dev/jetty/documentation/jetty-12/programming-guide/index.html

    [Online - Docs; ultima visita 16/03/2024];

  - Documentazione Guice:

    https://github.com/google/guice/wiki/Motivation

    [Online - Docs; ultima visita 16/03/2024];

  - Postman Docs:

    https://learning.postman.com/docs/introduction/overview/

    [Online - Docs; ultima visita 16/03/2024];

  - TestContainers Java Docs:

    https://java.testcontainers.org/

[Online - Docs; ultima visita 16/03/2024];

- Locust Docs:
  https://docs.locust.io/en/stable/
  [Online - Docs; ultima visita 16/03/2024];
- Docker Docs:
  https://docs.docker.com/
  [Online - Docs; ultima visita 16/03/2024];
- Mockito Docs:
  https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html
  [Online - Docs; ultima visita 16/03/2024];
- JUnit5 Guida:
  https://junit.org/junit5/docs/current/user-guide/
  [Online - Docs; ultima visita 16/03/2024];
- Hexagonal Architecture: Common pitfalls:
  https://medium.com/@albert.llousas/hexagonal-architecture-common-pitfalls-f155e12388a3
  [Online - Blog; ultima visita 05/04/2024];
- Hexagonal Architecture with Java – Tutorial:
  https://www.happycoders.eu/software-craftsmanship/hexagonal-architecture-java/
  [Online - Blog; ultima visita 05/04/2024];
- Hexagonal (Ports and Adapters) Architecture:
  https://medium.com/idealo-tech-blog/hexagonal-ports-adapters-architecture-e3617bcf00a0
  [Online - Blog; ultima visita 05/04/2024];
- Ready for changes with Hexagonal Architecture:

https://netflixtechblog.com/
ready-for-changes-with-hexagonal-architecture-b315ec967749
[Online - Blog; ultima visita 05/04/2024];

2 Tecnologie

Nella seguente sezione sono elencate e descritte le tecnologie impiegate nell'implementazione del prodotto richiesto nel capitolato. Lo scopo principale è assicurare che il software sia sviluppato impiegando le tecnologie più idonee in termini di efficienza, sicurezza e affidabilità.

2.1 Tecnologie per la programmazione

2.1.1 Linguaggi

Tabella 2: Linguaggi utilizzati per la codifica.

Nome	Versione	Descrizione
Java	21 LTS	Linguaggio di programmazione ad alto livello e orientato agli oggetti, noto per la sua portabilità, affidabilità e robustezza. Consigliato dall'azienda proponente.

2.1.2 Strumenti e servizi

Tabella 3: Strumenti e servizi utilizzati per la codifica.

Nome	Versione	Descrizione
Docker	26.0.0	Piattaforma di virtualizzazione leggera che consente di creare, distribuire e gestire applicazioni in contenitori software. I container Docker includono tutto il necessario per eseguire un'applicazione, tra cui codice, runtime, librerie di sistema e dipendenze.

Docker-compose	2.24.0	Strumento che semplifica la gestione di applicazioni Docker multi-container. Consente di definire e gestire facilmente l'intera infrastruttura dell'applicazione utilizzando un file di configurazione YAML.
Mongo DB	7.0	Database NoSQL orientato ai documenti, progettato per gestire grandi volumi di dati in modo flessibile e scalabile. Utilizza un modello di dati basato su documenti JSON, che consente una facile gestione e manipolazione dei dati senza uno schema rigido.
Maven	3.9.6	Strumento di gestione dei progetti software basato su Java. Automatizza il processo di compilazione, distribuzione e gestione delle dipendenze di un progetto.

2.1.3 Framework

Tabella 4: Framework utilizzati per la codifica.

Nome	Versione	Descrizione
Guice	7.0.0	Framework di dependency injection per Java. Guice facilita la gestione delle dipendenze all'interno delle applicazioni Java, migliorando la modularità e la manutenibilità del codice.

2.1.4 Librerie

Tabella 5: Librerie utilizzate per la codifica.

Nome	Versione	Descrizione

Jmap common	0.8.18	Libreria Java progettata per semplificare l'interazione con il protocollo JMAP. In questo modo è possibile creare, analizzare e manipolare i messaggi JMAP in modo efficiente e affidabile.
Jmap gson	0.8.18	Facilita la serializzazione delle entità definiti nella libreria *jmap-comon* che utilizza la libreria Gson, una libreria Java per la serializzazione e deserializzazione di oggetti JSON.
Jetty server	12.0.5	Server veloce e altamente scalabile, scritto in Java. È progettato per essere incorporato in altre applicazioni Java e offre un'implementazione completa dei protocolli HTTP e WebSocket.
Jetty util	12.0.5	Libreria di utilità sviluppata come parte del progetto Jetty. Questa libreria fornisce una serie di strumenti e funzioni utili per semplificare lo sviluppo e l'utilizzo di applicazioni Java, in particolare in contesti Web.
Guava	33.0.0	Libreria open-source scritta in Java. Essa fornisce un vasto insieme di utilità e strutture dati che vanno oltre le funzionalità disponibili nella libreria standard Java, offrendo soluzioni a molte esigenze comuni nello sviluppo di software.
Mongodb driver sync	4.11.1	Libreria fornita da MongoDB per consentire l'interazione sincrona con un database MongoDB utilizzando il linguaggio di programmazione Java. Questa libreria offre un'API intuitiva e facile da usare per eseguire operazioni CRUD.
SLF4J api	2.0.12	Libreria per la gestione dei log in Java che fornisce un'interfaccia semplice e unificata per l'utilizzo di vari framework di logging.

2.2 Tecnologie per l'analisi del codice

2.2.1 Analisi statica

Tabella 6: Tecnologie per l'analisi statica.

Nome	Versione	Descrizione
Compilatore Java	21 LTS	Traduce il codice sorgente Java in bytecode, un formato di file eseguibile interpretato dalla Java Virtual Machine. In questo modo si rende il codice Java altamente portabile.
Spotless	6.23.3	Strumento di formattazione del codice sorgente per progetti basati su JVM. È progettato per garantire uniformità e coerenza nel formato del codice all'interno di un progetto, facilitando la lettura, la manutenzione e la collaborazione.

2.2.2 Analisi dinamica

Tabella 7: Tecnologie per l'analisi dinamica.

Nome	Versione	Descrizione
Junit	5.9.3	Framework di test unitari per il linguaggio di programmazione Java. È utilizzato per scrivere, eseguire e organizzare test automatizzati per verificare il comportamento delle singole unità di codice all'interno di un'applicazione Java.
Mockito	5.1.1	Framework di mocking per il linguaggio di programmazione Java. È utilizzato principalmente per creare oggetti fittizi (mock) di classi e interfacce esistenti al fine di testare il comportamento di altre classi senza dipendere dalle implementazioni reali.

2.3 Tecnologie per i test

2.3.1 Linguaggi

Tabella 8: Linguaggi utilizzati per i test.

Nome	Versione	Descrizione
Python	3.12.2	Linguaggio di programmazione ad alto livello, interpretato e multiparadigma. Sintassi semplice e leggibile, che lo rende adatto sia ai principianti che agli sviluppatori esperti.

2.3.2 Framework

Tabella 9: Framework utilizzati per i test.

Nome	Versione	Descrizione
Locust	2.23.1	Framework open-source per il testing delle prestazioni e il carico degli applicativi. È progettato per simulare migliaia di utenti concorrenti interagendo con il sistema in questione, consentendo di valutare la capacità di resistenza e la scalabilità delle applicazioni e servizi.
Postman	10.22	Piattaforma di sviluppo API che consente agli sviluppatori di creare, testare, documentare e condividere API in modo efficiente.
TestContainers	10.22	È una libreria per il testing che facilita l'esecuzione di test automatizzati per le applicazioni che interagiscono con database, servizi di messaggistica e altri componenti esterni.

3 API

Le API$_G$ costituiscono il mezzo tramite il quale è possibile interagire con l'applicativo. In questa sezione verranno descritti gli endpoint accessibili e come interagire con essi. Quando un endpoint non viene trovato il server torna la risposta di default Error 404.

3.1 Endpoint

3.1.1 /.well-known/jmap

Endpoint standard definito nel draft ufficiale JMAP. É possibile interagirci tramite metodo GET, che restituirà esito positivo con codice HTTP 200 restituendo l'oggetto *Session Resource*$_G$ se trovato.

Descrizione	Codice HTTP	Payload
200 - Ok	La chiamata è di tipo GET e l'utente all'interno del campo di autenticazione nell'header della richiesta HTTP esiste nel sistema	*Session Resource* in formato JSON dell'utente richiesto
401 - Unauthorized	La chiamata è di tipo GET e l'autenticazione non è presente nell'header HTTP della richiesta, oppure l'utente fornito non esiste nel sistema	Ritorna la stringa "*Invalid Auth*"

Tabella 10: Riassunto dei codici restituiti da "/.well-known/jmap".

3.1.2 /api

Endpoint standard definito nel draft ufficiale, è possibile interagirci dopo aver eseguito l'autenticazione mediante metodo POST. L'autenticazione non è richiesta dal proponente, è quindi predisposto l'endpoint precedente come punto di accesso predefinito per l'applicativo.

Descrizione	Codice HTTP	Payload
200 - Ok	La chiamata è di tipo POST e le chiamate JMAP sono valide	*Method Responses* in formato JSON della chiamate inviate
500 - Internal Server Error	La chiamata è di tipo POST e il JSON nel payload della richiesta HTTP non segue le regole definite dallo standard JMAP	Ritorna la stringa JSON d'errore standard JMAP (sezione 3.1.3.1)

Tabella 11: Riassunto dei codici restituiti da "/api".

3.1.3 Invio delle chiamate JMAP

Le method call sono espresse in formato JSON e vengono inoltrate tramite metodo POST, il ritorno aspettato è una stringa JSON contenente le risposte alle richieste precedentemente inviate.

3.1.3.1 Errore Standard JMAP - JSON non valido

Quando una richiesta JMAP viene fatta al server, il JSON contente tutte le chiamate deve rispettare i seguenti requisiti:

- deve essere un JSON valido dal punto di vista sintattico;
- deve essere valido lo schema con cui vengono fatte le chiamate JMAP come da standard.

Nel caso in cui una di queste due regole viene meno il server ritorna come risposta il seguente messaggio d'errore, descritto nelle specifiche JMAP, in formato JSON:

```
{
"type":"urn:ietf:params:jmap:error:notJSON",
"status":500,
"detail":"See https://jmap.io/spec-core.html#errors"
}
```

4 Architettura

4.1 Architettura logica

Nel prodotto commissionato dal proponente la parte più importante è la business logic, in quanto deve assolvere al compito di elaborare e generare delle risposte alle chiamate JMAP che le arrivano. Tenendo anche in conto che non tutte le tecnologie usate sono quelle consigliate dall'azienda, e quindi potrebbe esserci il desiderio da parte del proponente di poter cambiare facilmente lo stack tecnologico. Inoltre considerando che il prodotto da sviluppare è una demo, potrebbe esserci la possibilità che possa venir modificato ulteriormente. Viste queste necessità il gruppo QB Software ha deciso di scegliere l'**architettura esagonale** per andare in contro a queste esigenze. L'architettura esagonale ci permette di separare le disponibilità in diversi componenti:

- *application logic* : dove viene implementata tutta la logica per permette al server di comunicare con l'esterno;
- *business logic* : dove viene implementata la parte più importante del prodotto. Questa parte è composta da quattro parti:
  - **porte**: rappresentano i punti d'ingresso o di uscita dalla business logic, le porte permettono di eliminare qualsiasi dipendenza della business logic dall'esterno. All'esterno della business logic ogni porta dovrà essere implementata con degli adapter;
  - **domain**: nel "domain" abbiamo tutte le entità di dominio definite all'interno della business logic, le quali non devono dipendere dalle parti esterne alla business logic;
  - **usecase**: definisco l'interfaccia (contratto) di come accedere alle funzionalità della business logic;
  - **service**: implementano gli usecase e permettono di implementare le funzionalità di business.
- *persistence logic*: implementa la logica di persistenza, nel nostro caso l'interazione con il database MongoDB.

L'architettura esagonale è stata scelta in quanto promuove una struttura robusta, chiara e flessibile consentendo di gestire al meglio la complessità dell'applicativo, conferendogli:

- **disaccoppiamento**: riduce l'accoppiamento tra le parti semplificando manutenzione ed evoluzione;
- **scalabilità**: facile estensione e modifica delle parti senza ripercussioni sulla business logic;
- **testabilità**: separazione tra dominio e porte, semplifica i test unitari e di integrazione.

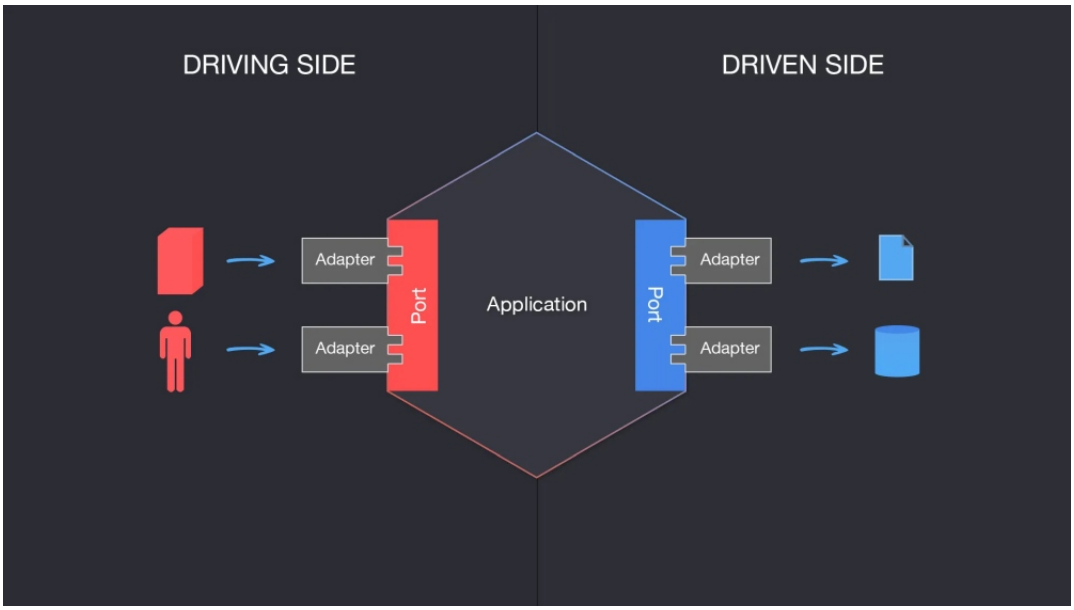

Figura 1: Esempio concettuale di architettura esagonale.

Assunto che il software realizzato sarà soggetto a future modifiche da parte del proponente, è di fondamentale importanza rendere le componenti dell'applicativo il più possibile indipendenti tra loro in modo che possano facilmente essere modificate o sostituite cercando di garantire flessibilità e longevità al prodotto finale nonostante le future evoluzioni.

Al centro dell'esagono risiede la business logic del server di posta elettronica, questa elabora le richieste in arrivo. In questa parte centrale risiedono le implementazioni delle funzionalità che eseguono e restituiscono gli esiti delle method call associate.

All'esterno dell'esagono, nel driving side, troviamo il chain of responsibility che fa in modo di dirottare le method call inviate dal client$_G$ al giusto controller, poi quest'ultimo avrà la responsabilità di delegare al service corretto l'elaborazione della risposta.

Invece lato driven side, troviamo le componenti per interfacciarsi al database MongoDB, l'implementazioni degli adapter per ogni porta, e altri strumenti utili a manipolare la persistenza di e-mail, cartelle, account$_G$, tracciare cambiamenti delle varie entità, ecc.

4.2 Architettura di deployment

4.2.1 Scelta e motivazioni

Dato il contesto dello sviluppo di un software finalizzato all'implementazione di un protocollo innovativo e alla valutazione delle sue prestazioni attraverso test di stress, si è optato per un'architettura che faccia utilizzo dei container. Questa decisione deriva da diversi fattori chiave.

In primo luogo, il software in questione non sarà messo veramente in produzione perché è una demo. Sarà utilizzato dal proponente per ulteriori analisi o per esaminare l'integrazione di specifiche funzioni di JMAP nei propri sistemi.

L'adozione di un'architettura che sfrutti i container è stata motivata dall'efficienza e facilità di implementazione. L'applicativo, avendo dimensioni contenute e finalità ben definite, permette al team di ridurre la complessità dello sviluppo, evitando le sfide tecniche associate alla gestione di un'architettura a microservizi.

Inoltre, considerando l'esperienza limitata del team di sviluppo e la necessità di frequenti modifiche al codice, una struttura che sfrutti i container si è rivelata più pratica. Tale struttura consente rapidità di sviluppo e una minore necessità di coordinamento tra servizi distinti, tipica invece delle architetture a microservizi.

In conclusione, sebbene le architetture a microservizi offrano vantaggi per applicazioni più complesse, come la scalabilità e l'aggiunta meno ostica di nuove funzionalità, la scelta dell'architettura deve anche tenere conto delle competenze del team di sviluppo e dei bisogni dell'azienda proponente Zextras. Il team QB Software, valutando i benefici rispetto agli sforzi richiesti, ha concluso che, per il progetto in corso, un'architettura costruita su container Docker è la soluzione più appropriata.

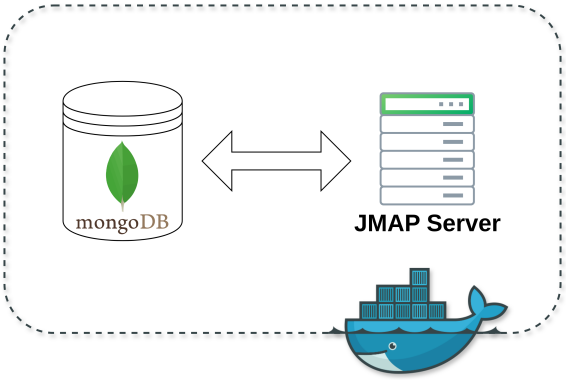

Figura 2: Architettura di deployment con docker-compose.

4.2.2 Uso di container

Docker ha stabilito uno standard industriale per l'uso dei container, questi sono ambienti isolati che contengono tutte le dipendenze e le impostazioni necessarie al corretto funzionamento dell'applicativo, facilitando notevolmente creazione, test e distribuzione.

I container condividono il kernel col sistema operativo, rendendoli efficienti e leggeri. A differenza delle macchine virtuali, i container isolano le applicazioni facendo sì che ogni container abbia il proprio namespace, rendendo le applicazioni di fatto più sicure.

L'utilizzo di Docker aiuta ad assicurare una certa portabilità in quanto software e configurazioni di supporto vengono impacchettati in immagini che possono essere eseguite coerentemente su diverse piattaforme. A seconda delle necessità è pure possibile avere container che si basano su sistemi operativi differenti.

La nostra configurazione Docker è definita nel file `docker-compose.yml`, essa ha lo scopo di gestire due container: web server e MongoDB.

1. **Jetty server:** è avviato mediante l'immagine *openjdk:21*, il servizio è esposto alla porta 9999 mappata all'omonima porta dell'host;

2. **Mongo database:** è avviato mediante l'immagine *mongo:3.8*, il servizio è esposto alla porta 27017 mappata all'omonima porta dell'host.

4.3 Database

4.3.1 Motivazioni

La scelta di MongoDB risiede in queste sue qualità:

- è un NoSQL, nato e pensato per i documenti JSON, questa caratteristica ci permette di salvare e recuperare in tutta semplicità i documenti che salviamo all'interno del database, eliminando così il problema di avere oggetti con dei campi sempre presenti (schema di base fisso) e alcuni opzionali (parte variabile dello schema del documento);
- è performante sulla ricerca per _id dei documenti, nel nostro prodotto i documenti vengono recuperati solo attraverso il campo _id del documento questo ci permette di eseguire query molto efficienti basate sull'indice principale di default di MongoDB. Questa scelta ha l'ulteriore vantaggio di eliminare la necessità di indici secondari che possono portare a occupare maggior memoria RAM del necessario. Inoltre il suo sistema di query ci permette di recuperare tutti i documenti appartenenti a un certo account attraverso il semplice campo _id;
- la semplicità con cui è possibile creare le collection ci permette di velocizzare i tempi per testare le soluzioni ipotizzate, e di valutarne l'adeguatezza senza spender troppo tempo nella definizione di uno schema come in un RDMS;
- la sua ottima e molto ricca documentazione ci ha permesso di capire il suo funzionamento, e di studiare le strategie migliori per salvare i dati nel database.

4.3.2 Collection

Di seguito tutte le collezioni progettate nel DB con il loro scopo:

- **Session**: contiene tutte le sessioni utente registrate;
- **account_state**: contiene tutti gli stati degli account registrati;
- **email**: contiene tutte le email$_G$ salvate nel sistema;
- **mailbox**: contiene tutte le mailbox$_G$ salvate nel sistema;
- **identity**: contiene tutte le identities$_G$ salvate nel sistema;
- **thread**: contiene tutte i thread$_G$ salvati nel sistema;
- **email_changes**: tiene tutti i documenti che tengono traccia dei cambiamenti dell'e-mail di ogni singolo account;

- **mailbox_changes**: tiene tutti i documenti che tengono traccia dei cambiamenti delle mailbox di ogni singolo account;
- **identity_changes**: tiene tutti i documenti che tengono traccia dei cambiamenti delle identities di ogni singolo account;
- **thread_changes**: tiene tutti i documenti che tengono traccia dei cambiamenti dei thread di ogni singolo account.

4.3.3 Design pattern

4.3.3.1 Inversion of control

In un progetto software di notevole dimensioni, la gestione e riduzione delle dipendenze diventano aspetti cruciali. Questa necessità si fa particolarmente sentire durante la manutenzione e aggiunta di nuove funzionalità quando il software è già in produzione. L'obiettivo del pattern in questione è quindi disaccoppiare il comportamento di un componente dalla risoluzione delle sue dipendenze. Se l'albero delle dipendenze si rivela essere una struttura eccessivamente complessa è facile ricadere in una situazione dove a una semplice modifica possa conseguire una serie di scomode modifiche a cascata sulle altre componenti coinvolte.

L'adozione del pattern permette di rispettare l'Hollywood principle secondo il quale un oggetto non dovrebbe chiamare esplicitamente i suoi collaboratori, ma dovrebbe piuttosto lasciare che qualcun altro, come un framework o un contenitore, si occupi di fornire le dipendenze.

Data questa considerazione, si è deciso di comune accordo di optare per il framework Guice. La scelta è qui ricaduta poiché oltre a essere una scelta popolare era già nota ad alcuni membri del team; esso permette di applicare il design pattern senza la necessità di definire inutili factories. Inoltre è possibile istanziare oggetti tramite un *injector* senza ricorrere alla keyword *new*.

Quando viene istanziato un *injector* vengono passati al costruttore di questo una serie di moduli ad hoc, in base al ruolo che esso deve svolgere, organizzati per feature come da best practice. Tramite questi moduli è possibile definire i bindings fra le interfacce e le loro implementazioni concrete, aggiungere parametri di configurazione per inizializzazioni specifiche. Per la parte di dominio si è optato di usare il metodo di provides di Guice per non inquinare la business logic.

Sono di fatto unità di configurazione che definiscono come le dipendenze interne un applicativo debbano esser fornite e gestite.

4.3.3.2 Chain of responsabilities (COR)

Questo è un pattern comportamentale che viene utilizzato per creare una catena di oggetti responsabili del trattamento di una richiesta. Viene formandosi una catena nella quale ogni oggetto ha la possibilità di gestire la richiesta o passarla al prossimo oggetto della catena.

Questo pattern risulta particolarmente utile in quanto un ipotetico dispatcher lato server che si occupa di ricevere le MethodCall dovrebbe trattare la richiesta a partire dalla determinazione del suo sottotipo, infatti nel momento di ricezione della MethodCall si conosce solo il supertipo di essa. La catena del COR è composta da *Handler* che controllano che la method call passata possa essere gestita, nel caso quest'ultima condizione sia vera allora l'handler prende la risposta e la passa al controller corretto, altrimenti passa al successivo handler. Nel caso non ci sia più nessun handler capace di gestire la richiesta allora il COR nel nostro caso tornerà un UnkownMethodCall.

In conclusione l'applicazione di questo pattern permette di rispettare al meglio il principio open/close e allo stesso tempo avere un risultato analogo a quello che si avrebbe con una cascata di if statements.

4.3.3.3 Adapter

Nel contesto di un'architettura esagonale, l'utilizzo di questo pattern è fondamentale, in quanto rende possibile mantenere indipendenza tra le componenti dell'applicativo senza compromettere l'integrazione delle componenti esterne. Così facendo si pone al centro la business logic e le sue funzionalità principali.

Ci sono due tipi di attori:

- **driving actors**: usano il domain per raggiungere il loro obiettivo;
- **driven actors**: mettono a disposizione le funzionalità necessarie per raggiungere l'obiettivo.

Un driven actor può essere un recipient che riceve informazioni dal domain oppure una repository che fornisce/riceve informazioni al/dal domain.

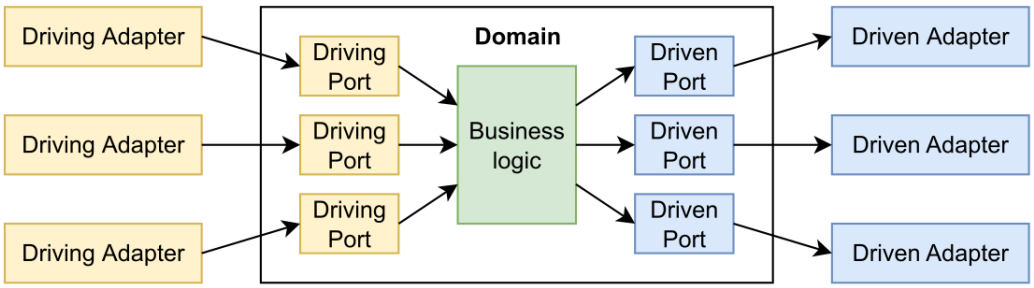

Figura 3: Esempio di struttura a porte e adapter

Un adapter utilizza l'ereditarietà per adattare l'interfaccia di una classe a un'altra. La classe adapter eredita dalla classe che si desidera adattare e implementa l'interfaccia richiesta dall'altra classe.

In sostanza permette a due componenti con interfacce diverse di comunicare senza che debbano essere modificate direttamente. Questo promuove il principio del "design per l'estensione, non per la modifica", rispettando al meglio il principio open/close, rendendo il codice più flessibile e manutenibile.

Sono state predisposte diverse porte di accesso (interfacce) tramite le quali le componenti esterne possono interagire col sistema. A ogni porta corrisponde un adapter che implementa l'interfaccia, questi vengono poi utilizzati per gestire la comunicazione con le componenti esterne.

4.3.3.4 Builder

È un design pattern creazionale molto utilizzato per creare oggetti complessi step by step. Esso risulta particolarmente utile quando si ha la necessità di creare oggetti con molte opzioni o parametri configurabili in modo flessibile, questo permette di creare diverse istanze dello stesso oggetto tramite lo stesso metodo di costruzione.

Abbiamo scelto di utilizzare questo pattern perché abbiamo molti oggetti che possono essere costruiti in diversi modi.

4.4 Diagramma delle classi

4.4.1 Introduzione

In questa sezione riportiamo il diagramma delle classi di ogni parte che compone il prodotto commissionato dal proponente, partendo da come vengono ascoltate le chiamate

ricevute agli endpoint dell'applicazione, per poi passare all'instradare la giusta chiamata JMAP, detta method call, verso il service di business che ne calcoli la risposta, e in fine l'implementazione delle classi di business che permettano di generare le risposte alla method call.

Di seguito introduciamo delle convenzioni per evitare di rendere il documento troppo prolisso:

- se in una classe che implementa il contratto di una interfaccia non specifichiamo i metodi ereditati dall'interfaccia stessa, questo significa che la classe implementa i metodi come da contratto dell'interfaccia e non sono necessarie ulteriori informazioni oltre a quella già fornite sulla descrizione del metodo nell'interfaccia;
- se non specificato il costruttore di una classe allora si assume che sia quello di default;
- se in una classe viene definita come "record" intendiamo dire che la classe funziona da contenitore di dati, ogni campo definito al suo interno è immutabile. Ogni campo ha un corrispettivo metodo, con identificativo uguale, che ritorna il campo stesso. Il costruttore è sotto inteso con tutti i campi definiti nel record. Esempio:

```
PointClass
Punto 2D nello spazio euclideo.

Campi/Metodi (record)
    - x: Double - Coordinata x;
    - y: Double - Coordinata y.

Equivalenza in Java:
public record PointClass(Double x, Double y) {}
```

- se tra i metodi di una classe viene scritto:

```
ConstructorByField
```

Costruttore.

allora significa che la classe ha un costruttore con tutti i campi indicati nella sezione attributi della classe stessa.

4.4.2 Application Logic

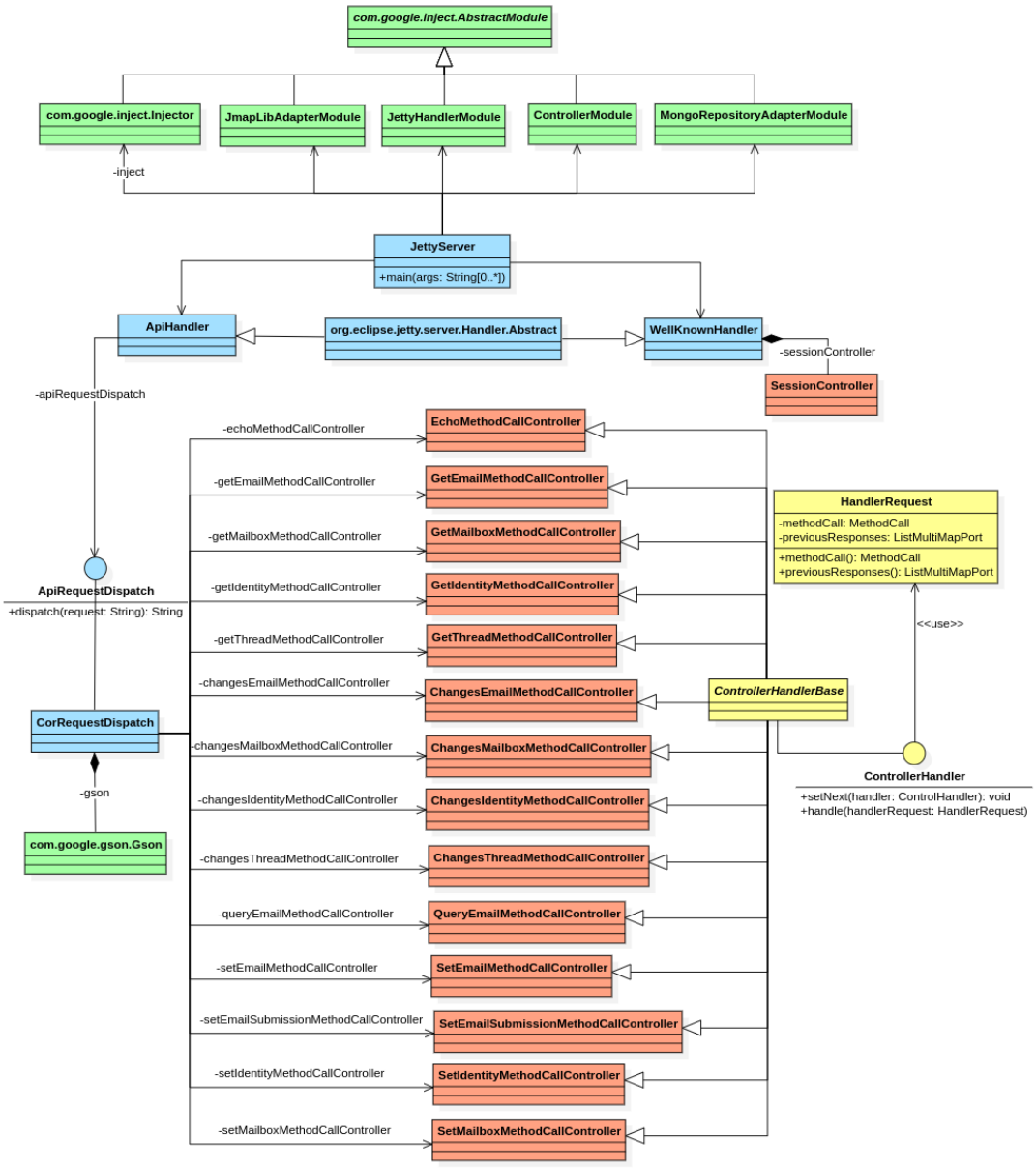

Figura 4: Diagramma delle classi che implementano l'application logic.

4.4.2.1 JettyServer

Main del programma, implementa ed esegue il server Jetty.

Metodi:

- `main(args: String[0..*])`

  Avvia il server e inietta le dipendenze all'interno degli handler.

4.4.2.2 JmapLibAdapterModule

Definisce i bindings delle porte della entità JMAP della business con il corrispettivo adapter, permettendo a Guice di risolvere le dipendenze riguardanti le porte del dominio.

4.4.2.3 JettyHandlerModule

Definisce i bindings degli handler di Jetty con le relative implementazioni, permettendo a Guice di risolvere le dipendenze riguardanti gli endpoint e configurazione varie per Jetty.

4.4.2.4 ControllerModule

Definisce dei metodi provides che permettono di fare la DI degli oggetti di dominio senza inquinare il dominio con le annotazioni di Guice, e dunque senza violare i principi dell'architettura esagonale.

4.4.2.5 MongoRepositoryAdapterModule

Definisce i bindings delle porte repository esposte dalla business con l'implementazione corrispettiva degli adapter per il database MongoDB.

4.4.2.6 ApiHandler

Implementa *Handler.Abstract* di Jetty, e mette a disposizione l'endpoint per ricevere le method call.

Attributi:

- `CONTEX_PATH: String`
  Context path dell'handler.
- `ENDPOINT_PATH: String`
  Endpoint dell'handler.
- `logger: org.slf4j.Logger`
  Logger di SLF4J.
- `apiRequestDispatch: ApiRequestDispatch`

Istanza che implementi un *ApiRequestDispatch* (sezione 4.4.2.8) che permette di instradare la chiamata verso il giusto algoritmo di risoluzione.

Metodi:

- `ApiHandler(apiRequestDispatch: ApiRequestDispatch)`
  Costruttore.
- `handle(request: Request, response: Response, callback: Callback): boolean`
  Implementa la logica per la gestione dell'endpoint.

4.4.2.7 WellKnownHandler

Implementa *Handler.Abstract* di Jetty, e mette a disposizione l'endpoint per ricevere una chiamata HTTP GET per recuperare la risorsa sessione dell'utente.

Attributi:

- `CONTEX_PATH: String`
  Context path dell'handler.
- `ENDPOINT_PATH: String`
  Endpoint dell'handler.
- `logger: org.slf4j.Logger`
  Logger di SLF4J.
- `sessionController: SessionController`
  Richiede il *SessionController* che serve ad accendere alla parte di business per la chiamata per recuperare la risorsa sessione dell'utente nel modo corretto.

Metodi:

- `WellKnownHandler(sessionController: SessionController)`
  Costruttore.
- `handle(request: Request, response: Response, callback: Callback): boolean`
  Implementa la logica per la gestione dell'endpoint.

4.4.2.8 ApiRequestDispatch

Interfaccia che dichiara il contratto che deve implementare la classe che implementi il routing della richiesta verso il giusto controller.

Metodi:

- `dispatch(request: String): String`
  Passa la richiesta al giusto controller e la ritorna la risposta in formato JSON.

4.4.2.9 CorRequestDispatch

Implementa l'interfaccia *ApiRequestDispatch* (paragrafo 4.4.2.8) attraverso il pattern Chain of Responsabilities.

Attributi

- `echoMethodCallController: EchoMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Echo.
- `getEmailMethodCallController: GetEmailMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Email/Get.
- `getIdentityMethodCallController: GetIdentitymethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Identity/Get.
- `getMailboxMethodCallController: GetMailboxMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Mailbox/Get.
- `getThreadMethodCallController: GetThreadMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Thread/Get.
- `changesEmailMethodCallController: ChangesEmailMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Email/Changes.

- `changesIdentityMethodCallController: ChangesIdentityMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Identity/Changes.
- `changesMailboxMethodCallController: ChangesMailboxMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Mailbox/Changes.
- `changesThreadMethodCallController: ChangesThreadMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Thread/Changes.
- `queryEmailMethodCallController: QueryEmailMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Email/Query.
- `setEmailMethodCallController: SetEmailMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Email/Set.
- `setIdentityMethodCallController: SetIdentityMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Identity/Set.
- `setsMailboxMethodCallController: SetMailboxMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP Mailbox/Set.
- `setEmailSubmissionMethodCallController: SetEmailSubmissionMethodCallController`
  Controller che permette di accedere alla logica di business per la chiamata JMAP EmailSubmission/Set.
- `gson: com.google.gson.Gson`
  Serializzatore/deserializzatore JSON configurato per la libreria JMAP.

4.4.2.10 Metodi:

- `ConstructorByField`
  Costruttore.
- `setupCor(): void`
  Collega gli handler del Chain Of Responsabilities.

4.4.2.11 HandlerRequest

Tiene le informazioni della richiesta, tra cui: la chiamata da fare e le risposte precedenti.

Attributi/Metodi (record):

- `methodCall: MethodCall`
  Chiamata JMAP da eseguire.
- `previousResponses: ListMultimapPort<String, ResponseInvocationPort>`
  Lista Multimap di Guava con tutte le risposte precedenti e riferiti al codice della methodcall che ha generato quella risposta.

4.4.2.12 ControllerHandler

Base per l'implementazione del pattern COR.

Metodi:

- `setNext(handler: ControllerHandler)`
  Permette di specificare qual è l'handler successivo che prova a gestire la richiesta nel caso questo non ci riesca.
- `handle(handlerRequest: HandlerRequest)`
  Richiesta passata all'handler.

4.4.2.13 ControllerHandlerBase

Implementa una base comune per tutti gli handler nel caso non esista nessun handler che riesce a gestire una richiesta, quest'ultimo ritorna un "UnkownMethodCall" error come da Standard JMAP.

4.4.2.14 SessionController

Permette di usare lo usecase della business *SessionUsecase* (paragrafo 4.4.3.2).

Attributi:

- `sessionUsecase: SessionUsecase`
  Usecase controllato.

4.4.2.15 EchoMethodCallController

Permette di usare lo usecase della business *EchoMethodCallUsecase* (paragrafo 4.4.4.2).

Attributi:

- `echoMethodCallUsecase: EchoMethodCallUsecase`
  Usecase controllato.

4.4.2.16 GetEmailMethodCallController

Permette di usare lo usecase della business *GetEmailMethodCallUsecase* (paragrafo 4.4.5.2).

Attributi:

- `getEmailMethodCallUsecase: GetEmailMethodCallUsecase`
  Usecase controllato.

4.4.2.17 GetMailboxMethodCallController

Permette di usare lo usecase della business *GetMailboxMethodCallUsecase* (paragrafo 4.4.7.2).

Attributi:

- `getMailboxMethodCallUsecase: GetMailboxMethodCallUsecase`
  Usecase controllato.

4.4.2.18 GetIdentityMethodCallController

Permette di usare lo usecase della business *GetIdentityMethodCallUsecase* (paragrafo 4.4.6.2).

Attributi:

- `getIdentitymethodCallUsecase: GetIdentityMethodCallUsecase`
  Usecase controllato.

4.4.2.19 GetThreadMethodCallController

Permette di usare lo usecase della business *GetThreadMethodCallUsecase* (paragrafo 4.4.8.2).

Attributi:

- `getThreadMethodCallUsecase: GetThreadMethodCallUsecase`
  Usecase controllato.

4.4.2.20 ChangesEmailMethodCallController

Permette di usare lo usecase della business *ChangesEmailMethodCallUsecase* (paragrafo 4.4.9.2).

Attributi:

- `changesEmailMethodCallUsecase: ChangesEmailMethodCallUsecase`
  Usecase controllato.

4.4.2.21 ChangesMailboxMethodCallController

Permette di usare lo usecase della business *ChangesMailboxMethodCallUsecase* (paragrafo 4.4.11.2).

Attributi:

- `changesMailboxMethodCallUsecase: ChangesMailboxMethodCallUsecase`
  Usecase controllato.

4.4.2.22 ChangesIdentityMethodCallController

Permette di usare lo usecase della business *ChangesIdentityMethodCallUsecase* (paragrafo 4.4.10.3).

Attributi:

- `changesIdentityMethodCallUsecase: ChangesIdentityMethodCallUsecase`
  Usecase controllato.

4.4.2.23 ChangesThreadMethodCallController

Permette di usare lo usecase della business *ChangesThreadMethodCallUsecase* (paragrafo 4.4.12.2).

Attributi:

- `changesThreadMethodCallUsecase: ChangesThreadMethodCallUsecase`
  Usecase controllato.

4.4.2.24 QueryEmailMethodCallController

Permette di usare lo usecase della business *QueryEmailMethodCallUsecase* (paragrafo 4.4.17.2).

Attributi:

- `queryEmailMethodCallUsecase: QueryEmailMethodCallUsecase`
  Usecase controllato.

4.4.2.25 SetEmailMethodCallController

Permette di usare lo usecase della business *SetEmailMethodCallUsecase* (paragrafo 4.4.13.2).

Attributi:

- `setEmailMethodCallUsecase: SetEmailMethodCallUsecase`
  Usecase controllato.

4.4.2.26 SetEmailSubmissionMethodCallController

Permette di usare lo usecase della business *SetEmailSubmissionMethodCallUsecase* (paragrafo 4.4.14.2).

Attributi:

- `setEmailSubmissionMethodCallUsecase: SetEmailSubmissionMethodCallUsecase`
  Usecase controllato.

4.4.2.27 SetMailboxMethodCallController

Permette di usare lo usecase della business *SetMailboxMethodCallUsecase* (paragrafo 4.4.16.2).

Attributi:

- `setMailboxMethodCallUsecase: SetMailboxMethodCallUsecase`
  Usecase controllato.

4.4.2.28 SetIdentityMethodCallController

Permette di usare lo usecase della business *SetIdentityMethodCallUsecase* (paragrafo 4.4.15.2).

Attributi:

- `setIdentityMethodCallUsecase: SetIdentityMethodCallUsecase`
  Usecase controllato.

4.4.3 SessionResource

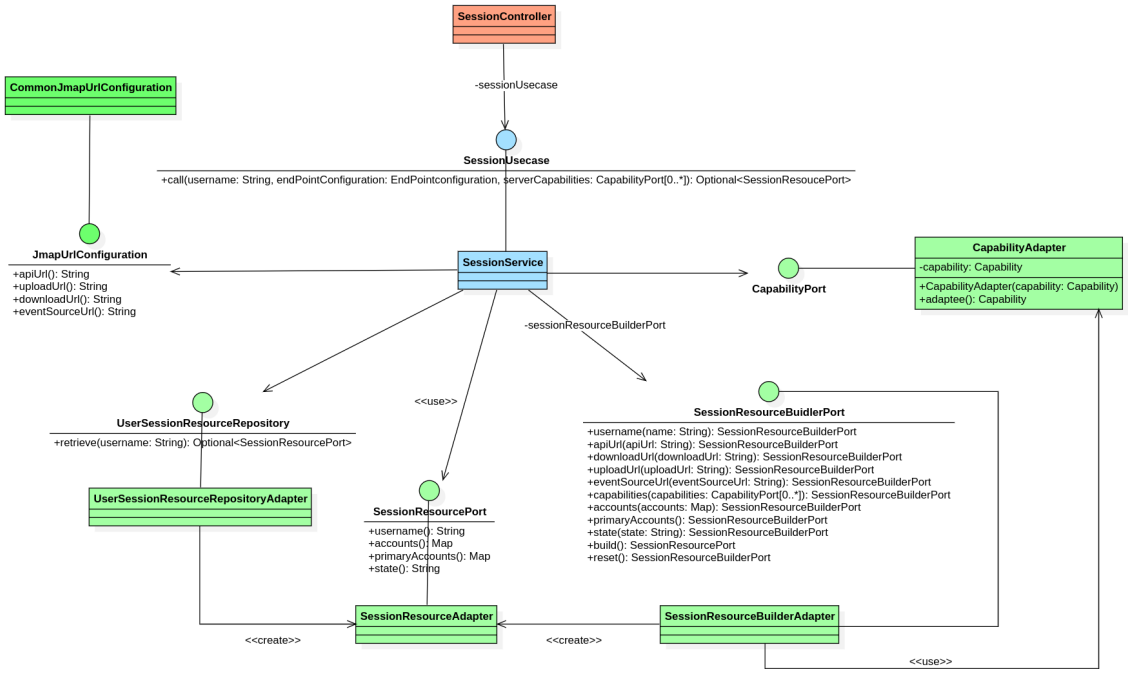

Figura 5: Diagramma delle classi che implementano per il recupero dell'oggetto Session Resource.

4.4.3.1 Riferimenti

- Per **SessionController** vedi la sezione 4.4.2.14.

4.4.3.2 SessionUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta dell'oggetto sessione per uno specifico utente e ritorna una risposta.

Metodi:

- `call(username:String, jmapUrlConfiguration: JmapUrlConfiguration, serverCapabilities:CapabilityPort[0..*]): Optional<SessionResoucePort>`

  Esegue la chiamata recuperando la sessione dell'utente dalla persistence logic e ritorna la risposta elaborata aggiornandola con le capabilities disponibili e gli URL configurati dalla parte application logic.

4.4.3.3 SessionService

Implementa l'interfaccia *SessionUsecase* (sezione 4.4.3.2).

Attributi:

- `sessionResourceBuilderPort: SessionResourceBuilderPort`
  Istanza di una porta che implementi il pattern Builder per l'entità *SessionResourcePort* (vedi sezione 4.4.18.7);
- `userSessionResourceRepository: UserSessionResourceRepository`
  Istanza di una porta che implementi l'adapter per accedere alla persistenza e permetta di recuperare la risorsa di sessione JMAP dell'utente specifica se esiste nel sistema.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.3.4 SessionResourceBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *SessionResourcePort* (vedi sezione 4.4.18.7).

Metodi:

- `username(name: String): SessionResourceBuilderPort`
  Permette di impostare lo username;
- `apiUrl(apiUrl: String): SessionResourceBuilderPort`
  Permette di impostare l'URL per ricevere le method call;
- `downloadUrl(downloadUrl: String): SessionResourceBuilderPort`
  Permette di impostare l'URL per scaricare dati grezzi;
- `uploadUrl(uploadUrl: String): SessionResourceBuilderPort`
  Permette di impostare l'URL per caricare dati grezzi;
- `eventSourceUrl(eventSourceUrl: String): SessionResourceBuilderPort`
  Permette di impostare l'URL dove collegarsi per ricevere ogni aggiornamento;

- `accounts(accounts: Map<String,AccountPort>): SessionResourceBuilderPort`
  Permette di indicare quali accounts appartengono all'utente o siano condivisi;
- `capabilities(capabilities: CapabilityPort[0..*]): SessionResourceBuilderPort`
  Permette di specificare quali funzionalità (capability) del server l'utente può usufruire;
- `primaryAccounts(primaryAccounts: Map<ClassAccountCapabilityPort,String>): SessionResourceBuilderPort`
  Permette di indicare quali sono gli account principali per ogni capability indicata precedentemente;
- `state(state: String): SessionResourceBuilderPort`
  Permette di indicare lo stato attuale della sessione all'ultima modifica;
- `build(): SessionResourcePort`
  Crea l'oggetto *SessionResoucePort* (vedi sezione 4.4.18.7);
- `reset(): SessionResourceBuilderPort`
  Esegue il reset del builder.

4.4.3.5 JmapUrlConfiguration

Dichiara il contratto che deve avere la classe che implementi l'insieme dei metodi che restituiscono l'URL per raggiungere ogni Url definito dallo standard JMAP.

4.4.3.6 Metodi:

- `apiUrl(): String`
  Ritorna l'URL per l'api dove inviare le method call;
- `uploadUrl(): String`
  Ritorna l'URL per il caricamento di dati grezzi sul server;
- `downloadUrl(): String`
  Ritorna l'URL per il download di dati grezzi dal server;

- `eventSourceUrl(): String`
  Ritorna l'URL per eventi generati dal server.

4.4.3.7 CommonJmapUrlConfiguration

Implementa l'interfaccia *JmapUrlConfiguration* (sezione 4.4.3.5) con tutti gli URL definiti nell'application logic.

4.4.3.8 UserSessionResourceRepository

Dichiara il contratto che deve implementare la classe che implementa la logica per recuperare la risorsa session di uno specifico utente.

Metodi:

- `retrieve(username: String): Optional<SessionResourcePort>`
  Recupera l'oggetto risorsa session dell'utente indicato, a patto che quest'ultimo esista.

4.4.3.9 UserSessionResourceRepositoryAdapter

Implementa l'interfaccia *UserSessionResourceRepository* (sezione 4.4.3.8).

Attributi:

- `connection: MongoConnection`
  Connessione al database MongoDB;
- `gson: Gson`
  Classe per la serializzazione/deserializzazione dei dati JSON, deve essere già configurata per l'entità JMAP e quelle del dominio;
- `COLLECTION: String`
  Stringa statica e costante per indicare qual è il nome della collection che contiene i dati che deve gestire l'adapter.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.4 EchoMethodCall

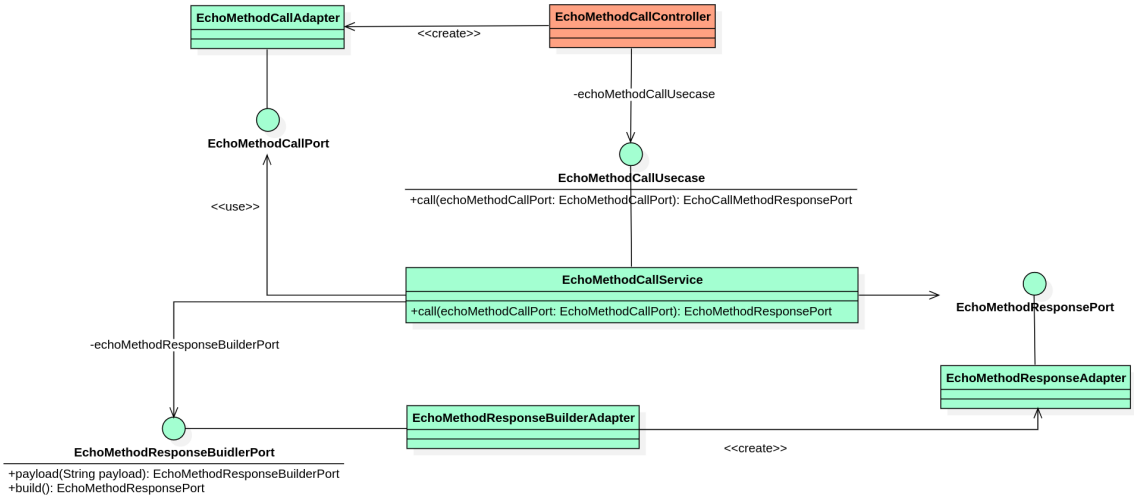

Figura 6: Diagramma delle classi che implementano la funzionalità /Echo.

4.4.4.1 Riferimenti

- Per **EchoMethodCallController** vedi la sezione 4.4.2.15.

4.4.4.2 EchoMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Echo e genera la corrispettiva risposta.

Metodi:

- `call(echoMethodCallPort: EchoMethodCallPort): EchoMethodResponsePort`
  Esegue la chiamata recuperando e generando una risposta identica alla chiamata per indicare che il server è nella rete.

4.4.4.3 EchoMethodCallService

Implementa l'interfaccia SessionUsecase (sezione 4.4.4.2).

Attributi:

- `echoMethodResponseBuilderPort: EchoMethodResponseBuilderPort`
  Builder per costruire un *EchoMethodResponsePort* (vedi sezione 4.4.4.6).

Metodi:

- `EchoMethodCallService(echoMethodResponseBuilderPort: EchoMethodResponseBuilderPort)`
  - Costruttore.

4.4.4.4 EchoMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *EchoMethodResponsePort* (vedi sezione 4.4.4.6).

Metodi:

- `payload(username: String): EchoMethodResponseBuilderPort`
  Permette di indicare il payload della risposta Echo;
- `build(): EchoMethodResponsePort`
  Crea l'oggetto *EchoMethodResponsePort* (vedi sezione 4.4.4.6);
- `reset(): EchoMethodResponseBuilderPort`
  Pulisce ogni campo del builder.

4.4.4.5 EchoMethodResponseBuilderAdapter

Implementa l'interfaccia *EchoMethodResponseBuilderPort* (sezione 4.4.4.4).

Attributi:

- `echoMethodResponseBuilder: rs.ltt.jmap.EchoMethodResponseBuilder`
  Builder della libreria JMAP.

4.4.4.6 EchoMethodResponsePort

Interfaccia che permette di ritornare la risposta alla method call echo.

4.4.4.7 EchoMethodResponseAdapter

Implementazione del *EchoMethodResponsePort* (vedi paragrafo 4.4.4.6).

Attributi:

- `echoMethodResponse: EchoMethodResponse`

  EchoMethodResponse (adaptee) della libreria JMAP adatto.

Metodi:

- `ConstructorByField`

  Costruttore.

- `adaptee(): EchoMEthodResponse`

  Ritorna l'adaptee.

4.4.5 GetEmailMethodCall

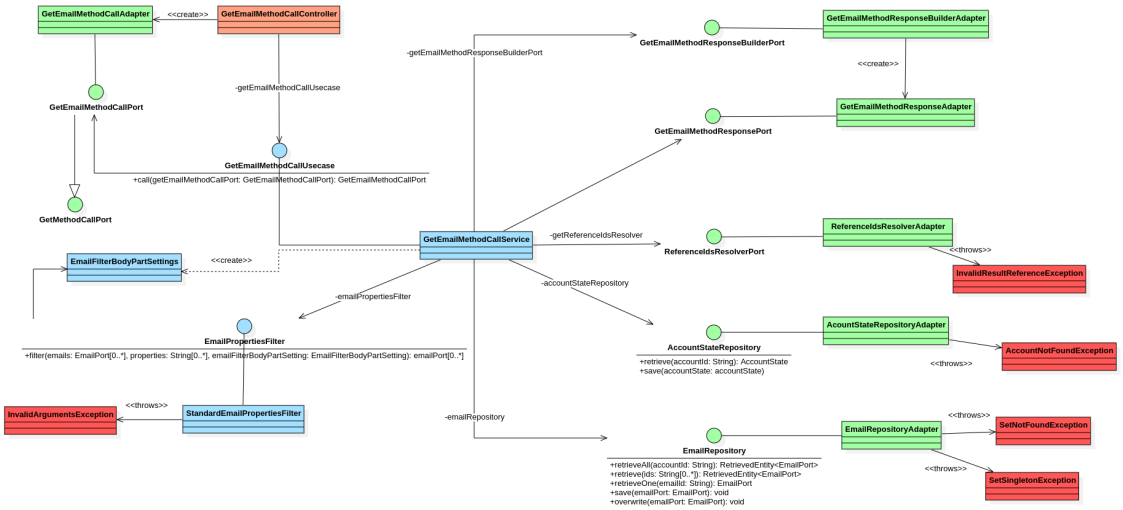

Figura 7: Diagramma delle classi che implementano la Email/Get.

4.4.5.1 Riferimenti

- Per **GetEmailMethodCallController** vedi la sezione 4.4.2.16;
- per **GetMethodCallPort** vedi la sezione 4.4.18.28;
- per **ReferenceIdsResolverPort** vedi la sezione 4.4.18.16;
- per **ReferenceIdsResolverAdapter** vedi la sezione 4.4.18.17;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;

- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **EmailRepository** vedi la sezione 4.4.19.3;
- per **EmailRepositoryAdapter** vedi la sezione 4.4.19.4;
- per **eccezioni** vedi la sezione 4.4.20.

4.4.5.2 GetEmailMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la chiamata della porta *GetEmailMethodCallPort* (sezione 4.4.5.7) e ritorna una risposta.

Metodi:

- `call(getEmailMethodCallPort: GetEmailMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): GetEmailMethodCallPort`
  Esegue la chiamata *getEmailMethodCallPort* passata e ritorna la risposta elaborata.

4.4.5.3 GetEmailMethodCallService

Implementa l'interfaccia *GetEmailMethodCallUsecase* (paragrafo 4.4.5.2) .

Attributi:

- `getEmailMethodResponseBuilderPort: GetEmailMethodResponseBuidlerPort`
  Istanza di una porta che implementi il pattern Builder per *GetEmailMethodResponsePort*.
- `accountStateRepository: AccountStateRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1).
- `emailRepository: EmailRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *EmailPort* (paragrafo 4.4.18.2) .

- `ReferenceIdsResolverPort: ReferenceIdsResolverPort`
  Istanza di una porta che permetta di risolvere gli ids temporanei passati dal client con i corrispettivi creati dal server.

- `emailPropertiesFilter: EmailPropertiesFilter`
  Istanza di una porta che permetta di filtrare l'e-mail recuperate in base ad alcuni campi che vengono passati al filtro, quest ultimo deve restituire tutte l'e-mail con solo quei campi.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.5.4 EmailFilterBodyPartSettings

Classe che tiene alcune impostazioni per il filtraggio dei campi di un e-mail da passare all'oggetto che implementi l'interfaccia *EmailPropertiesFilter* (sezione 4.4.5.5).

Attributi/Metodi (record)

- `bodyProperties: String[0..*]`
  Lista di proprietà da recuperare per ogni *EmailBodyPartPort* (sezione 4.4.18.4).

- `fecthTextBodyValues: Boolean`
  Se vero ogni parte dell'e-mail che include la proprietà "text/*" nel campo *textBody* va inclusa nel risultato del filtraggio.

- `fetchHtmlBodyValues: Boolean`
  Se vero ogni parte dell'e-mail che include la proprietà "text/*" nel campo *htmlBody* va inclusa nel risultato del filtraggio.

- `fecthAllBodyValyes: Boolean`
  Se vero ogni parte dell'e-mail che include la proprietà "text/*" nel campo *bodyStructure* dell'e-mail va inclusa nel risultato del filtraggio.

- `maxBodyValuesBytes: Long`
  Se > 0 allora la somma di ogni bodyValues deve pesare meno di questa variabile.

4.4.5.5 EmailPropertiesFilter

Dichiara il contratto che deve implementare la classe che filtra l'e-mail in base alle proprietà passate.

Metodi:

- `filter(in emails:EmailPort[0..*], in properties:String[0..*], in emailFilterBodyPartSetting:EmailFilterBodyPartSetting): emailPort[0..*]`
  Per ogni e-mail all'interno dell'array passato al metodo vengono restituiti in un array le stesse e-mail ma solo con i campi specificati nel argomento properties.

4.4.5.6 StandardEmailPropertiesFilter

Implementa l'interfaccia *EmailPropertiesFilter* (sezione 4.4.5.5), come da Standard JMAP.

Metodi:

- `emailFilter(emailPort: EmailPort, properties: String[0..*]): EmailPort`
  Filtra le proprietà dell'e-mail passata in base a quelle indicate nel parametro argomento *properties*, e ritorna l'e-mail solo con i campi richiesti e l'id sempre incluso.

4.4.5.7 GetEmailMethodCallPort

Dichiara l'interfaccia che deve avere un oggetto che rappresenti la chiamata JMAP Email/Get ed estende l'interfaccia più generale per le /Get method call: *GetMethodCallPort* (paragrafo 4.4.18.28).

Metodi:

- `getBodyProperties(): String[0..*]`
- `getFetchTextBodyValues(): Boolean`
- Se vero vanno recuperati tutti i *BodyValues* che hanno come proprietà "text/*" nel campo *textBody*.

- `getFetchHtmlBodyValues(): Boolean`
  Se vero vanno recuperati tutti i *BodyValues* che hanno come proprietà "text/*" nel campo *htmlBody*.
- `getFetchAllBodyValues(): Boolean`
  Se vero vanno recuperati tutti i *BodyValues* definiti nel campo *BodyStructure*.
- `getMaxBodyValuesBytes(): Long`
  Grandezza massima in Byte dei valori del corpo dell'e-mail da ritornare.

4.4.5.8 GetEmailMethodCallAdapter

Implementa la porta *GetEmailMethodCallPort* (paragrafo 4.4.5.7).

Attributi:

- `getEmailMethodCall: GetEmailMethodCall`
  Adaptee dell'adapter, è la classe che rappresenta la Email/Get method call nella libreria JMAP.

Metodi:

- `ConstructorByField`
  Costruttore.
- `adaptee(): GetEmailMethodCall`
  ritorna l'adaptee.

4.4.5.9 GetEmailMethodResponsePort

Definisce l'interfaccia che deve avere l'adapter per la risposta a una Email/Get method call.

4.4.5.10 GetEmailMethodResponseAdapter

Implementa l'interfaccia *GetEmailMethodResponsePort* (paragrafo 4.4.5.9).

Attributi:

- `getEmailMethodResponse: GetEmailMethodResponse`
  Adaptee dell'adapter, è la classe che rappresenta la Email/Get method call nella libreria JMAP.

Metodi:

- `adaptee(): GetEmailMethodResponse`
  Ritorna l'adaptee.

4.4.6 GetIdentityMethodCall

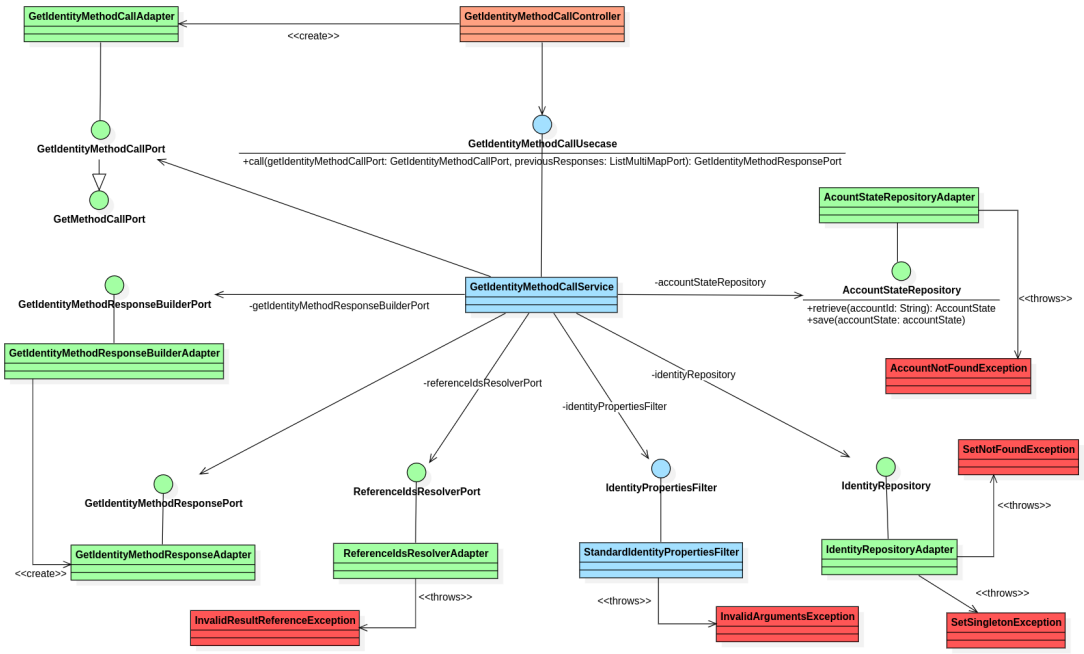

Figura 8: Diagramma delle classi che implementano la Identity/Get.

4.4.6.1 Riferimenti

- Per **GetIdentityMethodCallController** vedi la sezione 4.4.2.18;
- per **GetMethodCallPort** vedi la sezione 4.4.18.28;
- per **ReferenceIdsResolverPort** vedi la sezione 4.4.18.16;
- per **ReferenceIdsResolverAdapter** vedi la sezione 4.4.18.17;

- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **IdentityRepository** vedi la sezione 4.4.19.7;
- per **IdentityRepositoryAdapter** vedi la sezione 4.4.19.8;
- per **eccezioni** vedi la sezione 4.4.20.

4.4.6.2 GetIdentityMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Identity/Get e genera la corrispettiva risposta.

Metodi:

- `call(getIdentityMethodCallPort: GetIdentityMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): EchoMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutte l'entità Identity richieste.

4.4.6.3 GetIdentityMethodCallService

Implementa l'interfaccia *GetIdentityMethodCallUsecase* (sezione 4.4.6.2).

Attributi:

- `getIdentityMethodResponseBuilderPort: GetIdentityMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per *GetIdentityMethodResponsePort*;
- `accountStateRepository: AccountStateRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `identityRepository: IdentityRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *IdentityRepository* (paragrafo 4.4.19.7) ;

- `ReferenceIdsResolverPort: ReferenceIdsResolverPort`
  Istanza di una porta che permetta di risolvere gli ids temporanei passati dal client con i corrispettivi creati dal server;
- `identityPropertiesFilter: IdentityPropertiesFilter`
  Istanza di una porta che permetta di filtrare le identity recuperate in base ad alcuni campi che vengono passati al filtro, quest'ultimo deve restituire tutte le identità con solo quei campi.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.6.4 IdentityPropertiesFilter

Definisce l'interfaccia che deve avere una classe per implementare il filtro per l'entità identity.

Metodi:

- `filter(identityPorts: IdentityPort, properties: String[0..*]): IdentityPort`
  Filtra l'entità *IdentityPort* (paragrafo 4.4.18.18) in base agli attributi passati.

4.4.6.5 StandardIdentityPropertiesFilter

Implementa l'interfaccia *IdentityPropertiesFilter* (paragrafo 4.4.6.4) come da standard JMAP.

Attributi:

- `identityBuilderPort: identityBuilderPort`
  Builder per costruire entità *IdentityPort*.

Metodi:

- `filterIdentity(identityPort: IdentityPort, properties: String[0..*])`
  Filtra una entità *IdentityPort* in base alle properties passate come argomento, come da standard JMAP.

4.4.6.6 GetIdentityMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *GetIdentityMethodResponsePort* (vedi sezione 4.4.6.8).

Metodi:

- `list(identityPorts: IdentityPort[0..*]): GetIdentityMethodResponseBuilderPort`
  Lista delle identities recuperate e filtrate in modo corretto;
- `notFound(identityIdsNotFound: String[0..*]): GetIdentityMethodResponseBuilderPort`
  Lista degli ids di ogni identity non trovata tra quelle richieste dalla chiamata;
- `state(state: String): GetIdentityMethodResponseBuilderPort`
  Stato che il client raggiunge dopo aver salvato le identities recuperate;
- `build(): GetIdentityMethodResponsePort`
  Crea l'oggetto *GetIdentityMethodResponsePort* (vedi sezione 4.4.6.8);
- `reset(): GetIdentityMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.6.7 GetIdentityMethodResponseBuilderAdapter

Implementa l'interfaccia *GetIdentityMethodResponseBuilderPort* (sezione 4.4.6.6).

Attributi:

- `getIdentityMethodResponseBuilder: GetIdentityMethodResponseBuilder`
  Adaptee dell'adapter, è il builder della libreria JMAP per la risposta alle chiamate JMAP Identity/Get.

4.4.6.8 GetIdentityMethodResponsePort

Interfaccia che dichiara il contratto che l'adapter che la implementa deve rispettare per rappresentare la risposta a una chiamata Identity/Get.

4.4.6.9 GetIdentityMethodResponseAdapter

Implementa l'interfaccia *GetIdentityMethodResponsePort* (paragrafo 4.4.6.8).

Attributi:

- `getIdentityMethodResponse: GetIdentityMethodResponse`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una risposta alla chiamta JMAP Identity/Get.

Metodi:

- `ConstructorByField`
  Costruttore.
- `adaptee(): GetIdentityMethodResponse`
  Ritorna l'adaptee().

4.4.6.10 GetIdentityMethodCallPort

Definisce il contratto che deve implementare l'adapter per rappresentare una chiamata Identity/Get. Estende l'interfaccia di base *GetMethodCallPort* (paragrafo 4.4.18.28).

4.4.6.11 GetIdentityMethodCallAdapter

Attributi:

- `getIdentityMethodCall: GetIdentityMethodCall`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una chiamata JMAP Identity/Get.

Metodi:

- `ConstructorByField`
  Costruttore.
- `adaptee(): GetIdentityMethodCall`
  Ritorna l'adaptee().

4.4.7 GetMailboxMethodCall

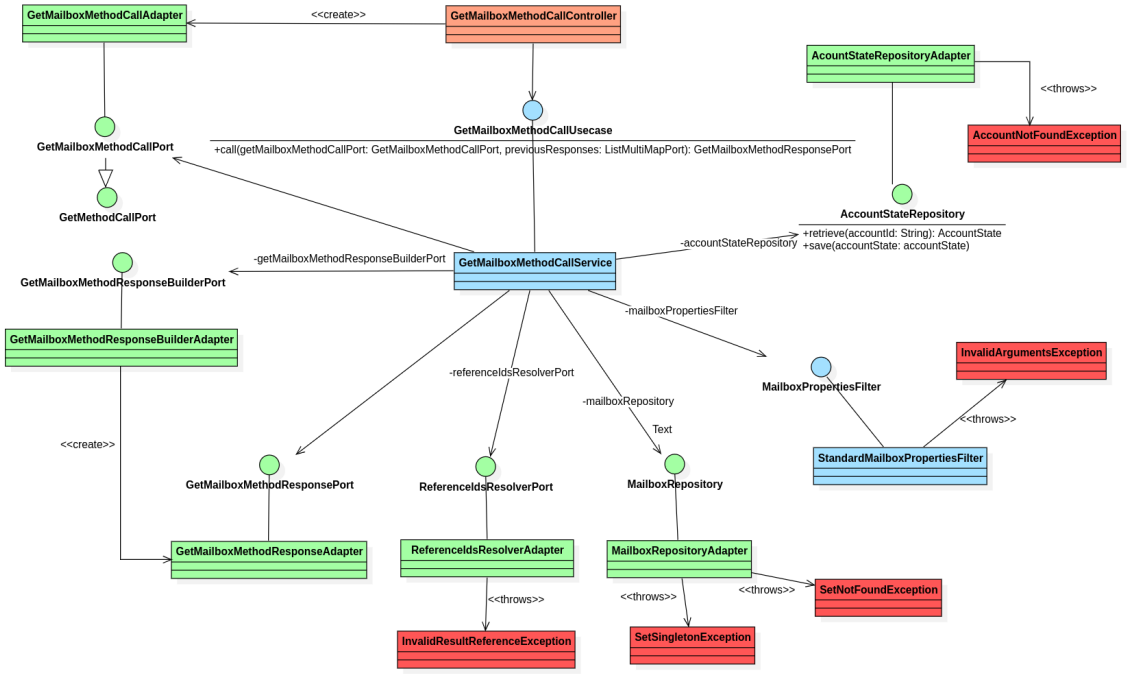

Figura 9: Diagramma delle classi che implementano la Mailbox/Get.

4.4.7.1 Riferimenti

- Per **GetMailboxMethodCallController** vedi la sezione 4.4.2.17;
- per **GetMethodCallPort** vedi la sezione 4.4.18.28;
- per **ReferenceIdsResolverPort** vedi la sezione 4.4.18.16;
- per **ReferenceIdsResolverAdapter** vedi la sezione 4.4.18.17;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **MailboxRepository** vedi la sezione 4.4.19.5;
- per **MailboxRepositoryAdapter** vedi la sezione 4.4.19.6;
- per **eccezioni** vedi la sezione 4.4.20.

4.4.7.2 GetMailboxMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Mailbox/Get e genera la corrispettiva risposta.

Metodi:

- `call(getMailboxMethodCallPort: GetMailboxMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): GetMailboxMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutte le entità mailbox richieste.

4.4.7.3 MailboxPropertiesFilter

Definisce l'interfaccia che deve avere un filtro per filtrare le proprietà di una entità mailbox.

Metodi:

- `filter(mailboxPorts: MailboxPort[0..*], properties: String[0..*]): MailboxPort[0..*]`
  Ritorna le *MailboxPort* (paragrafo 4.4.18.6) filtrate per le proprietà passate nell'argomento properties.

4.4.7.4 StandardMailboxPropertiesFilter

Implementa l'interfaccia *MailboxPropertiesFilter* (paragrafo 4.4.7.3), implementando il filtro come da standard JMAP.

Attributi:

- `mailboxBuilderPort: MailboxBuilderPort`
  Builder per gli oggetti *MailboxPort*.

Metodi:

- `mailboxFilter(mailboxPort: MailboxPort, properties: String[0..*]): MailboxPort`
  Filtra le proprietà di una singola mailbox come da standard JMAP.

4.4.7.5 GetMailboxMethodCallService

Implementa l'interfaccia *GetMailboxMethodCallUsecase* (sezione 4.4.7.2).

Attributi:

- `getMailboxMethodResponseBuilderPort: GetMailboxMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per *GetMailboxMethodResponsePort* (paragrafo 4.4.7.8) ;
- `accountStateRepository: AccountStateRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `identityRepository: IdentityRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *MailboxPort* (sezione 4.4.18.6);
- `ReferenceIdsResolverPort: ReferenceIdsResolverPort`
  Istanza di una porta che permetta di risolvere gli ids temporanei passati dal client con i corrispettivi creati dal server;
- `mailboxPropertiesFilter: MailboxPropertiesFilter`
  Istanza di una porta che permetta di filtrare le mailbox recuperate in base ad alcuni campi che vengono passati al filtro, quest'ultimo deve restituire tutte le mailbox con solo quei campi.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.7.6 GetMailboxMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *GetMailboxMethodResponsePort* (vedi sezione 4.4.7.8).

Metodi:

- `list(mailboxPorts: MailboxPort[0..*]): GetMailboxMethodResponseBuilderPort`
  Lista delle mailbox recuperate e filtrate in modo corretto;
- `notFound(mailboxIdsNotFound: String[0..*]): GetMailboxMethodResponseBuilderPort`
  Lista degli ids di ogni mailbox non trovate tra quelle richieste dalla chiamata;
- `state(state: String): GetMailboxMethodResponseBuilderPort`
  Stato che il client raggiunge dopo aver salvato le mailbox recuperate;
- `build(): GetMailboxMethodResponsePort`
  Crea l'oggetto *GetMailboxMethodResponsePort* (vedi sezione 4.4.7.8);
- `reset(): GetMailboxMethodResponseBuilderPort`
  Fa il reset del builder.

4.4.7.7 GetMailboxMethodResponseBuilderAdapter

Implementa l'interfaccia *GetMailboxMethodResponseBuilderPort* (sezione 4.4.7.6).

Attributi:

- `getMailboxMethodResponseBuilder: getMailboxMethodResponseBuilder`
  Adaptee dell'adapter, builder per la risposta Mailbox/Get della libreria JMAP.

4.4.7.8 GetMailboxMethodResponsePort

Definisce il contratto che devono implementare gli adapter che vogliano rappresentare una risposta alla chiamata Mailbox/Get.

4.4.7.9 GetMailboxMethodResponseAdapter

Implementa l'interfaccia *GetMailboxMethodResponsePort* (paragrafo 4.4.7.8) .

Attributi:

- `getMailboxMethodResponse: GetMailboxMethodResponse`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una risposta alla chiamata Mailbox/Get.

Metodi:

- `adaptee(): GetMailboxMethodResponse`
  Ritorna l'adaptee.

4.4.7.10 GetMailboxMethodCallPort

Definisce il contratto che deve implementare l'oggetto che rappresenta una chiamata JMAP Mailbox/Get, estende l'interfaccia *GetMethodCallPort* (paragrafo 4.4.18.28).

4.4.7.11 GetMailboxMethodCallAdapter

Implementa la porta *GetMailboxMethodCallPort* (paragrafo 4.4.7.10).

Attributi:

- `getMailboxMethodCall: GetMailboxMethodCall`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una chiamata JMAP Mailbox/Get.

Metodi:

- `adaptee(): GetMailboxMethodCall`
  Ritorna l'adaptee.

4.4.8 GetThreadMethodCall

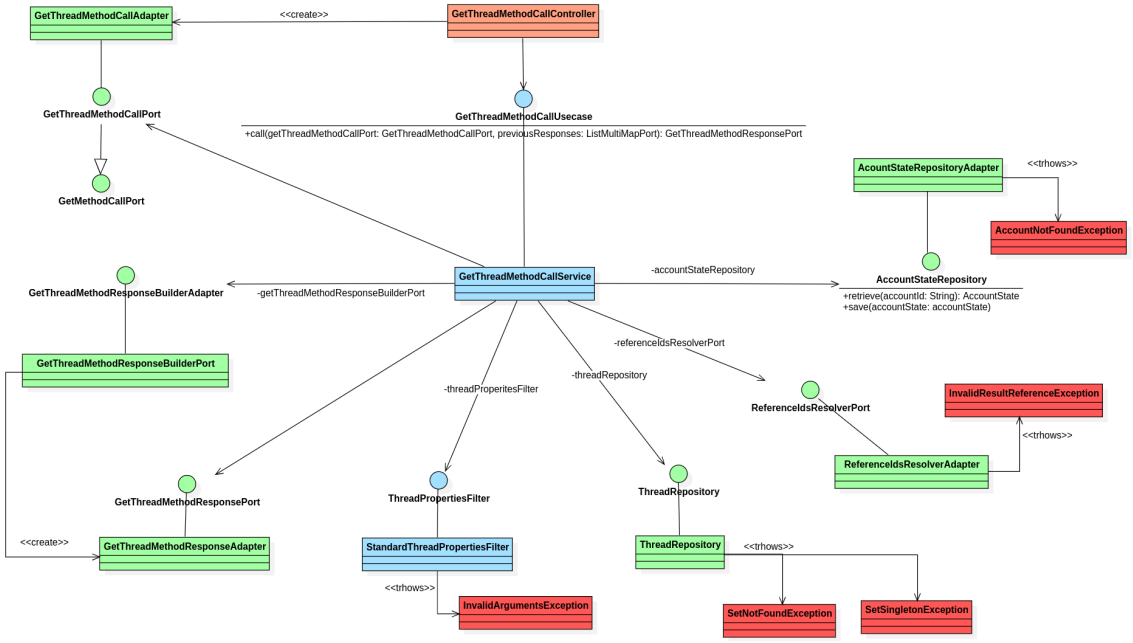

Figura 10: Diagramma delle classi che implementano la Thread/Get.

4.4.8.1 Riferimenti

- Per **GetThreadMethodCallController** vedi la sezione 4.4.2.19;
- per **GetMethodCallPort** vedi la sezione 4.4.18.28;
- per **ReferenceIdsResolverPort** vedi la sezione 4.4.18.16;
- per **ReferenceIdsResolverAdapter** vedi la sezione 4.4.18.17;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **ThreadRepository** vedi la sezione 4.4.19.9;
- per **ThreadRepositoryAdapter** vedi la sezione 4.4.19.10;
- per **eccezioni** vedi la sezione 4.4.20.

4.4.8.2 GetThreadMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Thread/Get e genera la corrispettiva risposta.

Metodi:

- `call(getThreadMethodCallPort: GetThreadMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): GetThreadMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutte le entità thread richieste.

4.4.8.3 GetThreadMethodCallService

Implementa l'interfaccia *GetThreadMethodCallUsecase* (sezione 4.4.8.2).

Attributi:

- `getThreadMethodResponseBuilderPort: GetThreadMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per *GetThreadMethodResponsePort*;
- `accountStateRepository: AccountStateRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `identityRepository: IdentityRepository`
  Istanza di una porta che permetta di recuperare dalla persistenza l'oggetto *ThreadPort* (sezione 4.4.18.19);
- `ReferenceIdsResolverPort: ReferenceIdsResolverPort`
  Istanza di una porta che permetta di risolvere gli ids temporanei passati dal client con i corrispettivi creati dal server;
- `threadPropertiesFilter: ThreadPropertiesFilter`
  Istanza di una porta che permetta di filtrare i thread recuperati in base ad alcuni campi che vengono passati al filtro, quest'ultimo deve restituire tutti i thread con solo quei campi.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.8.4 ThreadPropertiesFilter

Interfaccia che definisce il contratto che deve avere la classe che implementa il filtro per l'entità thread.

Metodi:

- `filter(threadPorts: ThreadPort[0..*], properties: String[0..*]): ThreadPort[0..*]`
  Filtra tutti gli oggetti threadPorts in base alle properties passate come argomento, ritorna le entità filtrate.

4.4.8.5 StandardThreadPropertiesFilter

Implementa l'interfaccia *ThreadPropertiesFilter* (paragrafo 4.4.8.4).

Attributi:

- `threadBuilderPort: ThreadBuilderPort`
  Builder per l'entità *ThreadPort* (paragrafo 4.4.18.19).

Metodi:

- `ConstructorByField`
  Costruttore.
- `filterThread(threadPort: ThreadPort, properties: String[0..*]: ThreadPort)`
  Filtra una singola *ThreadPort* in base alle proprietà fornite e la restituisce con i campi filtrati.

4.4.8.6 GetThreadMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *GetThreadMethodResponsePort* (vedi sezione 4.4.8.8).

Metodi:

- `list(threadPorts: ThreadPort[0..*]): GetThreadMethodResponseBuilderPort`
  Lista dei thread recuperati e filtrati in modo corretto;
- `notFound(threadIdsNotFound: String[0..*]): GetThreadMethodResponseBuilderPort`
  Lista degli ids di ogni thread non trovato tra quelli richiesti dalla chiamata;
- `state(state: String): GetThreadMethodResponseBuilderPort`
  Stato che il client raggiunge dopo aver salvato i thread recuperati;
- `build(): GetThreadMethodResponsePort`
  Crea l'oggetto *GetThreadMethodResponsePort* (vedi sezione 4.4.8.8);
- `reset(): GetThreadMethodResponseBuilderPort`
  Pulisce ogni campo del builder.

4.4.8.7 GetThreadMethodResponseBuilderAdapter

Implementa l'interfaccia *GetThreadMethodResponseBuilderPort* (sezione 4.4.8.6).

Attributi:

- `getThreadMethodResponseBuilder: GetThreadMethodResponseBuilder`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una risposta alla chiamata JMAP Thread/Get.

4.4.8.8 GetThreadMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che rappresenta una risposta alla chiamata JMAP Thread/Get.

4.4.8.9 GetThreadMethodResponseAdapter

Implementa l'interfaccia *GetThreadMethodResponsePort* (paragrafo 4.4.8.8).

Attributi:

- `getThreadMethodResponse: GetThreadMethodResponse`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una risposta alla chiamata JMAP Thread/Get.

Metodi:

- `ConstructorByField`
  Costruttore.
- `adaptee(): GetThreadMethodResponse`
  Ritorna l'adaptee.

4.4.8.10 GetThreadMethodCallPort

Interfaccia che definisce il contratto che l'adapter deve implementare per rappresentare una chiamata JMAP Thread/Get. Estende l'interfaccia di base *GetMethodCallPort* (paragrafo 4.4.18.28).

4.4.8.11 GetThreadMethodCallAdapter

Implementa l'interfaccia *GetThreadMethodCallPort* (paragrafo 4.4.8.10).

Attributi:

- `getThreadMethodCall: GetThreadMethodCall`
  Adaptee dell'adapter, oggetto della libreria JMAP che rappresenta una chiamata JMAP Thread/Get.

Metodi:

- `ConstructorByField`
  Costruttore.
- `adaptee(): GetThreadMethodCall`
  Ritorna l'adaptee.

4.4.9 ChangesEmailMethodCall

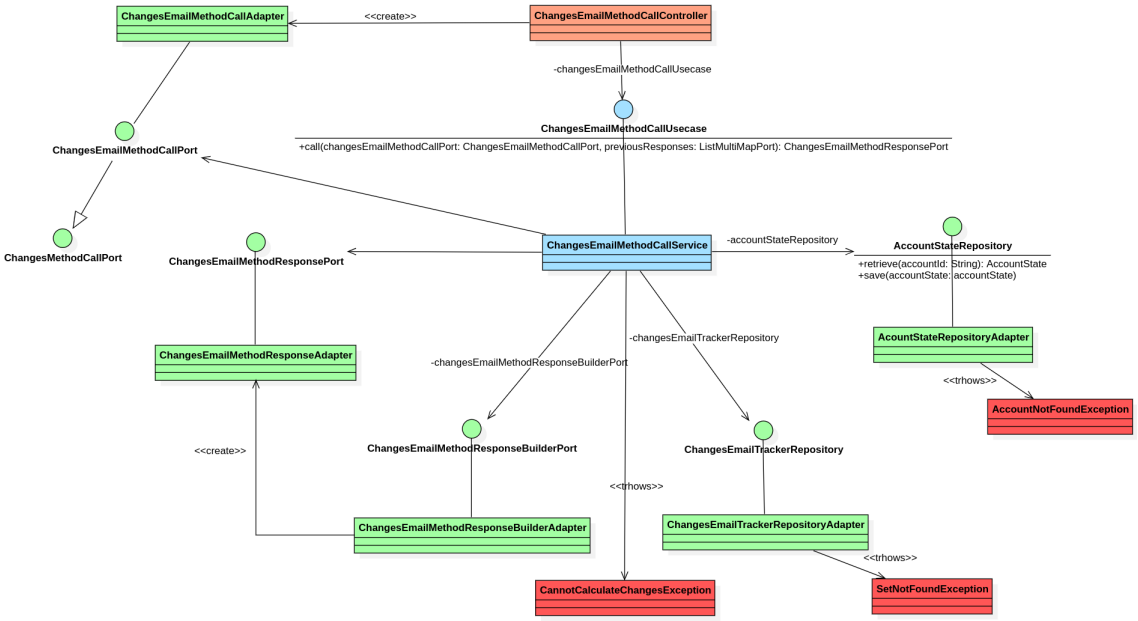

Figura 11: Diagramma delle classi che implementano la Email/Changes.

4.4.9.1 Riferimenti

- Per **ChangesEmailMethodCallController** vedi la sezione 4.4.2.20;
- per **ChangesMethodCallPort** vedi la sezione 4.4.18.29;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **EmailChangesTrackerRepository** vedi la sezione 4.4.19.11;
- per **EmailChangesTrackerRepositoryAdapter** vedi la sezione 4.4.19.12;
- per la definizione delle **eccezioni** vedi la sezione 4.4.20.

4.4.9.2 ChangesEmailMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Email/Changes e genera la corrispettiva risposta dove vengono riportati tutti i cambiamenti alle email a partire dallo stato fornito dal client durante la chiamata.

Metodi:

- `call(changesEmailMethodCallPort: ChangesEmailMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): ChangesEmailMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutti i cambiamenti che hanno subito le email dallo stato fornito se è possibile calcolarlo.

4.4.9.3 ChangesEmailMethodCallService

Implementa l'interfaccia *ChangesEmailMethodCallUsecase* (paragrafo 4.4.9.2).

Attributi:

- `changesEmailMethodResponseBuilderPort: changesEmailMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per l'entità *ChangesEmailMethodResponsePort* (paragrafo 4.4.9.6);
- `accountStateRepository: AccountStateRepository`
  Istanza della porta che permette di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `emailChangesTrackerRepository: EmailChangesTrackerRepository`
  Porta che permette di recuperare dalla persistenza una specifica entità che implementi *EmailChangesTracker* (paragrafo 4.4.18.8).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.9.4 ChangesEmailMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *ChangesEmailMethodResponsePort* (vedi sezione 4.4.9.6).

Metodi:

- `accountId(accountId: String): ChangesEmailMethodResponseBuilderPort`
  Id dell'account di cui si stanno ritornando le modifiche;
- `oldState(oldState: String): ChangesEmailMethodResponseBuilderPort`
  Stato del client prima di applicare tutti i cambiamenti ritornati, ovvero lo stato inviato durante la chiamata;
- `newState(newState: String): ChangesEmailMethodResponseBuilderPort`
  Nuovo stato del client dopo aver applicato tutti i cambiamenti presenti nella risposta alla chiamata;
- `hasMoreChanges(hasMoreChanges: Boolean): ChangesEmailMethodResponseBuilderPort`
  Notifica il client che i cambiamenti ritornati non sono tutti, ma solo una parte;
- `created(ids: String[0..*]): ChangesEmailMethodResponseBuilderPort`
  Id delle email create dall'ultimo stato del client fino a quello attuale del server;
- `updated(ids: String[0..*]): ChangesEmailMethodResponseBuilderPort`
  Id delle email modificate dall'ultimo stato del client fino a quello attuale del server;
- `destroyed(ids: String[0..*]): ChangesEmailMethodResponseBuilderPort`
  Id delle email distrutte dall'ultimo stato del client fino a quello attuale del server;
- `build(): ChangesEmailMethodResponsePort`
  Crea l'oggetto *ChangesEmailMethodResponsePort* (vedi sezione 4.4.9.6);
- `reset(): GetThreadMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.9.5 ChangesEmailMethodResponseBuilderAdapter

Implementa l'interfaccia *ChangesEmailMethodResponseBuilderPort* (sezione 4.4.9.4).

Attributi:

- `changesEmailMethodResponseBuilder: ChangesEmailMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che permette di usare il builder sulla risposta alla chiamata JMAP Email/Changes.

4.4.9.6 ChangesEmailMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Email/Changes.

4.4.9.7 ChangesEmailMethodResponseAdapter

Implementa l'interfaccia *ChangesEmailMethodResponsePort* (paragrafo 4.4.9.6).

Attributi:

- `changesEmailMethodResponse: ChangesEmailMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Email/Changes.

Metodi:

- `adaptee(): ChangesEmailMethodResponse`
  Ritorna l'adaptee.

4.4.9.8 ChangesEmailMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Email/Changes. Estende l'interfaccia di base *ChangesMethodCallPort* (paragrafo 4.4.18.29).

4.4.9.9 ChangesEmailMethodCallAdapter

Implementa l'interfaccia *ChangesEmailMethodCallPort* (paragrafo 4.4.9.8).

Attributi:

- `changesEmailMethodCall: ChangesEmailMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Email/Changes.

Metodi:

- `adaptee(): ChangesEmailMethodCall`

  Ritorna l'adaptee.

4.4.10 ChangesIdentityMethodCall

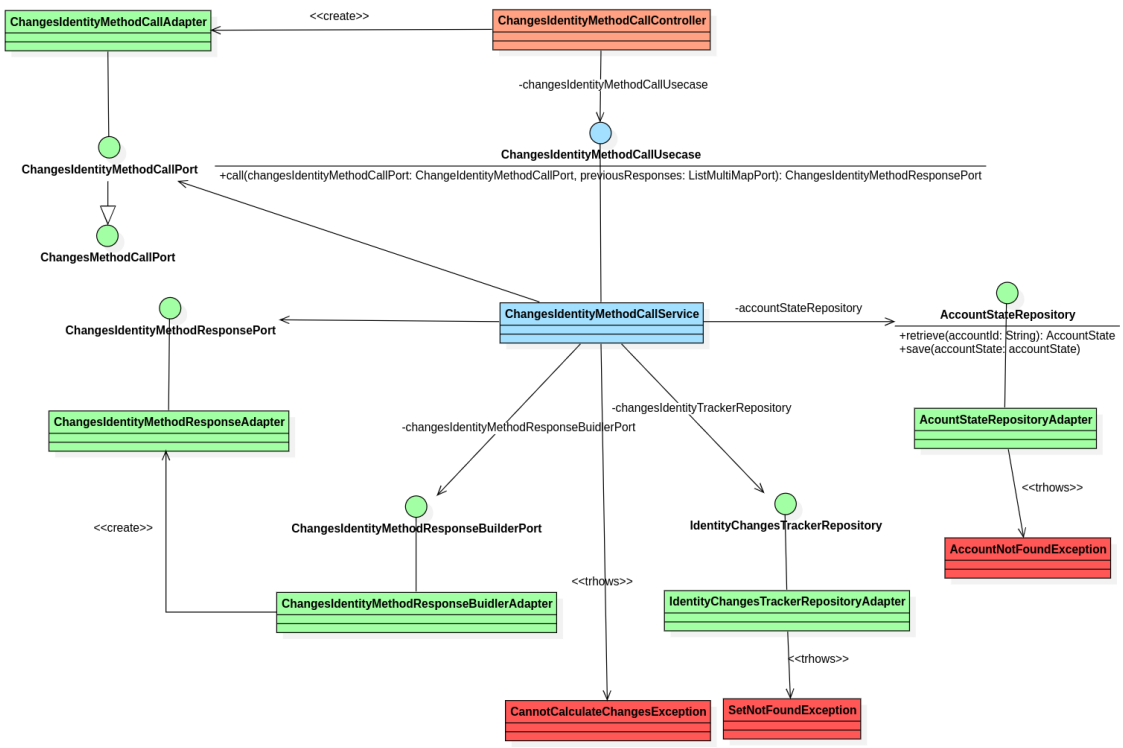

Figura 12: Diagramma delle classi che implementano la Identity/Changes.

4.4.10.1 Riferimenti

- Per **ChangesIdentityMethodCallController** vedi la paragrafo 4.4.2.22;
- per **ChangesMethodCallPort** vedi la paragrafo 4.4.18.29;
- per **AccountStateRepository** vedi la paragrafo 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la paragrafo 4.4.19.2;
- per **IdentityChangesTrackerRepository** vedi la paragrafo 4.4.19.15;
- per **IdentityChangesTrackerRepositoryAdapter** vedi la paragrafo 4.4.19.16;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.10.2 ChangesIdentityMethodCallUsecase

4.4.10.3 ChangesIdentityMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Identity/Changes e genera la corrispettiva risposta dove vengono riportati tutti i cambiamenti alle identities a partire dallo stato fornito dal client durante la chiamata.

Metodi:

- `call(changesIdentityMethodCallPort: ChangesIdentityMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): ChangesIdentityMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutti i cambiamenti che hanno subito le identities dallo stato fornito se è possibile calcolarlo.

4.4.10.4 ChangesIdentityMethodCallService

Implementa l'interfaccia *ChangesIdentityMethodCallUsecase* (paragrafo 4.4.10.3).

Attributi:

- `changesIdentityMethodResponseBuilderPort: changesIdentityMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per l'entità *ChangesIdentityMethodResponsePort* (paragrafo 4.4.10.7);
- `accountStateRepository: AccountStateRepository`
  Istanza della porta che permette di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `identityChangesTrackerRepository: IdentityChangesTrackerRepository`
  Porta che permette di recuperare dalla persistenza una specifica entità che implementi *IdentityChangesTracker* (paragrafo 4.4.18.12).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.10.5 ChangesIdentityMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *ChangesIdentityMethodResponsePort* (vedi paragrafo 4.4.10.7).

Metodi:

- `accountId(accountId: String): ChangesIdentityMethodResponseBuilderPort`
  Id dell'account di cui si stanno ritornando le modifiche;
- `oldState(oldState: String): ChangesIdentityMethodResponseBuilderPort`
  Stato del client prima di applicare tutti i cambiamenti ritornati, ovvero lo stato inviato durante la chiamata;
- `newState(newState: String): ChangesIdentityMethodResponseBuilderPort`
  Nuovo stato del client dopo aver applicato tutti i cambiamenti presenti nella risposta alla chiamata;
- `hasMoreChanges(hasMoreChanges: Boolean): ChangesIdentityMethodResponseBuilderPort`
  Notifica il client che i cambiamenti ritornati non sono tutti, ma solo una parte;
- `created(ids: String[0..*]): ChangesIdentityMethodResponseBuilderPort`
  Id delle identities create dall'ultimo stato del client fino a quello attuale del server;
- `updated(ids: String[0..*]): ChangesIdentityMethodResponseBuilderPort`
  Id delle identities modificate dall'ultimo stato del client fino a quello attuale del server;
- `destroyed(ids: String[0..*]): ChangesIdentityMethodResponseBuilderPort`
  Id delle identities distrutte dall'ultimo stato del client fino a quello attuale del server;
- `build(): ChangesIdentityMethodResponsePort`
  Crea l'oggetto *ChangesIdentityMethodResponsePort* (paragrafo 4.4.10.7);

- `reset(): GetThreadMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.10.6 ChangesIdentityMethodResponseBuilderAdapter

Implementa l'interfaccia *ChangesIdentityMethodResponseBuilderPort* (paragrafo 4.4.10.5).

Attributi:

- `changesIdentityMethodResponseBuilder: ChangesIdentityMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Identity/Changes.

4.4.10.7 ChangesIdentityMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Identity/Changes.

4.4.10.8 ChangesIdentityMethodResponseAdapter

Implementa l'interfaccia *ChangesIdentityMethodResponsePort* (paragrafo 4.4.10.7).

Attributi:

- `changesIdentityMethodResponse: ChangesIdentityMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Identity/Changes.

Metodi:

- `adaptee(): ChangesIdentityMethodResponse`
  Ritorna l'adaptee.

4.4.10.9 ChangesIdentityMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Identity/Changes. Estende l'interfaccia di base *ChangesMethodCallPort* (paragrafo 4.4.18.29).

4.4.10.10 ChangesIdentityMethodCallAdapter

Implementa l'interfaccia *ChangesIdentityMethodCallPort* (paragrafo 4.4.10.9).

Attributi:

- `changesIdentityMethodCall: ChangesIdentityMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Identity/Changes.

Metodi:

- `adaptee(): ChangesIdentityMethodCall`
  Ritorna l'adaptee.

4.4.11 ChangesMailboxMethodCall

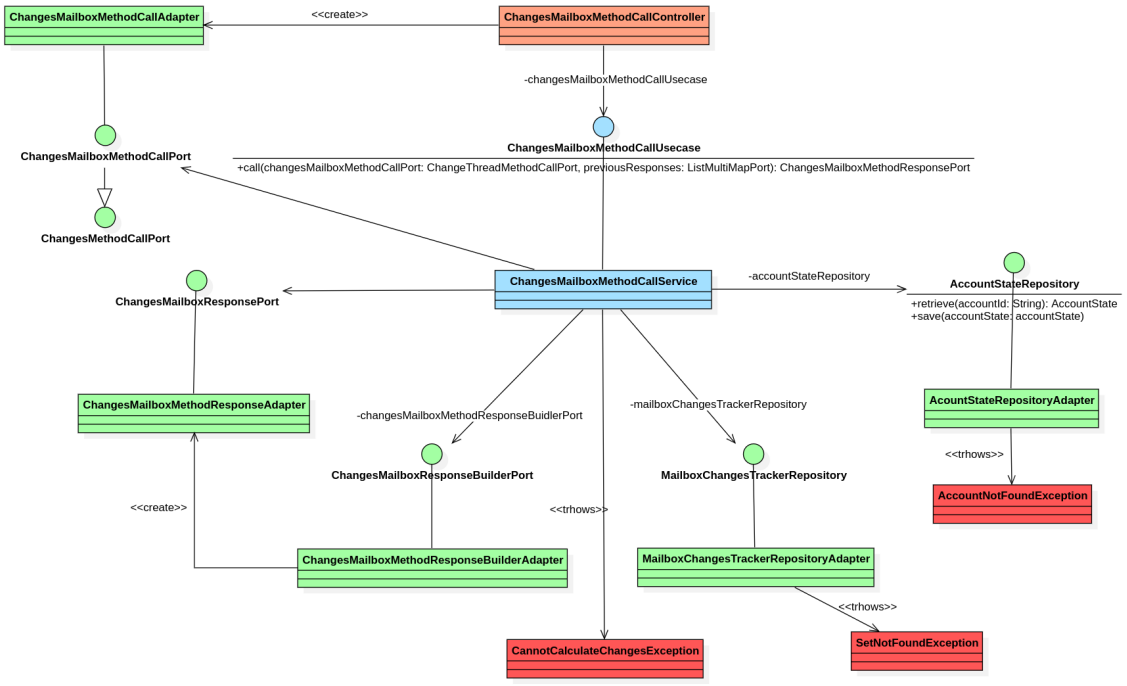

Figura 13: Diagramma delle classi che implementano la Mailbox/Changes.

4.4.11.1 Riferimenti

- Per **ChangesMailboxMethodCallController** vedi la paragrafo 4.4.2.21;
- per **ChangesMethodCallPort** vedi la paragrafo 4.4.18.29;

- per **AccountStateRepository** vedi la paragrafo 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la paragrafo 4.4.19.2;
- per **MailboxChangesTrackerRepository** vedi la paragrafo 4.4.19.13;
- per **MailboxChangesTrackerRepositoryAdapter** vedi la paragrafo 4.4.19.14;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.11.2 ChangesMailboxMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Mailbox/Changes e genera la corrispettiva risposta dove vengono riportati tutti i cambiamenti alle mailbox a partire dallo stato fornito dal client durante la chiamata.

Metodi:

- `call(changesMailboxMethodCallPort: ChangesMailboxMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): ChangesMailboxMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutti i cambiamenti che hanno subito le mailbox dallo stato fornito se è possibile calcolarlo.

4.4.11.3 ChangesMailboxMethodCallService

Implementa l'interfaccia *ChangesMailboxMethodCallUsecase* (paragrafo 4.4.11.2).

Attributi:

- `changesMailboxMethodResponseBuilderPort: changesMailboxMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per l'entità *ChangesMailboxMethodResponsePort* (paragrafo 4.4.11.6);
- `accountStateRepository: AccountStateRepository`
  Istanza della porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (paragrafo 4.4.18.1);

- `mailboxChangesTrackerRepository: MailboxChangesTrackerRepository`
  Porta che permette di recuperare dalla persistenza una specifica entità che implementi *MailboxChangesTracker* (paragrafo 4.4.18.10).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.11.4 ChangesMailboxMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *ChangesMailboxMethodResponsePort* (vedi paragrafo 4.4.11.6).

Metodi:

- `accountId(accountId: String): ChangesMailboxMethodResponseBuilderPort`
  Id dell'account di cui si stanno ritornando le modifiche;
- `oldState(oldState: String): ChangesMailboxMethodResponseBuilderPort`
  Stato del client prima di applicare tutti i cambiamenti ritornati, ovvero lo stato inviato durante la chiamata;
- `newState(newState: String): ChangesMailboxMethodResponseBuilderPort`
  Nuovo stato del client dopo aver applicato tutti i cambiamenti presenti nella risposta alla chiamata;
- `hasMoreChanges(hasMoreChanges: Boolean): ChangesMailboxMethodResponseBuilderPort`
  Notifica il client che i cambiamenti ritornati non sono tutti, ma solo una parte;
- `created(ids: String[0..*]): ChangesMailboxMethodResponseBuilderPort`
  Id delle mailbox create dall'ultimo stato del client fino a quello attuale del server;
- `updated(ids: String[0..*]): ChangesMailboxMethodResponseBuilderPort`
  Id delle mailbox modificate dall'ultimo stato del client fino a quello attuale del server;

- `destroyed(ids: String[0..*]): ChangesMailboxMethodResponseBuilderPort`
  Id delle mailbox distrutte dall'ultimo stato del client fino a quello attuale del server;
- `build(): ChangesMailboxMethodResponsePort`
  Crea l'oggetto *ChangesMailboxMethodResponsePort* (vedi paragrafo 4.4.11.6);
- `reset(): GetThreadMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.11.5 ChangesMailboxMethodResponseBuilderAdapter

Implementa l'interfaccia *ChangesMailboxMethodResponseBuilderPort* (paragrafo 4.4.11.4).

Attributi:

- `changesMailboxMethodResponseBuilder: ChangesMailboxMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Mailbox/Changes.

4.4.11.6 ChangesMailboxMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Mailbox/Changes.

4.4.11.7 ChangesMailboxMethodResponseAdapter

Implementa l'interfaccia *ChangesMailboxMethodResponsePort* (paragrafo 4.4.11.6).

Attributi:

- `changesMailboxMethodResponse: ChangesMailboxMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Mailbox/Changes.

Metodi:

- `adaptee(): ChangesMailboxMethodResponse`
  Ritorna l'adaptee.

4.4.11.8 ChangesMailboxMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Mailbox/Changes. Estende l'interfaccia di base *ChangesMethodCallPort* (paragrafo 4.4.18.29).

4.4.11.9 ChangesMailboxMethodCallAdapter

Implementa l'interfaccia *ChangesMailboxMethodCallPort* (paragrafo 4.4.11.8).

Attributi:

- `changesMailboxMethodCall: ChangesMailboxMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Mailbox/Changes.

Metodi:

- `adaptee(): ChangesMailboxMethodCall`
  Ritorna l'adaptee.

4.4.12 ChangesThreadMethodCall

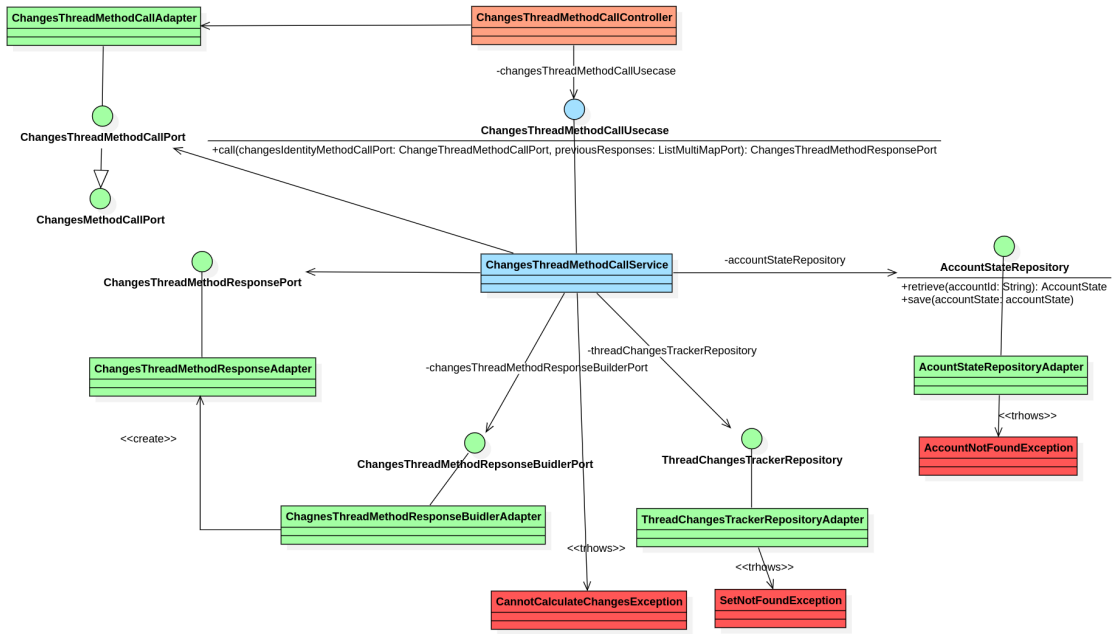

Figura 14: Diagramma delle classi che implementano la Thread/Changes.

4.4.12.1 Riferimenti

- Per **ChangesThreadMethodCallController** vedi la paragrafo 4.4.2.23;
- per **ChangesMethodCallPort** vedi la paragrafo 4.4.18.29;
- per **AccountStateRepository** vedi la paragrafo 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la paragrafo 4.4.19.2;
- per **ThreadChangesTrackerRepository** vedi la paragrafo 4.4.19.17;
- per **ThreadChangesTrackerRepositoryAdapter** vedi la paragrafo 4.4.19.18;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.12.2 ChangesThreadMethodCallUsecase

Dichiara il contratto che deve avere la classe che elabora la richiesta Thread/Changes e genera la corrispettiva risposta dove vengono riportati tutti i cambiamenti ai thread a partire dallo stato fornito dal client durante la chiamata.

Metodi:

- `call(changesThreadMethodCallPort: ChangesThreadMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): ChangesThreadMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna tutti i cambiamenti che hanno subito i thread dallo stato fornito se è possibile calcolarlo.

4.4.12.3 ChangesThreadMethodCallService

Implementa l'interfaccia *ChangesThreadMethodCallUsecase* (paragrafo 4.4.12.2).

Attributi:

- `changesThreadMethodResponseBuilderPort: changesThreadMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern Builder per l'entità *ChangesThreadMethodResponsePort* (paragrafo 4.4.12.7);
- `accountStateRepository: AccountStateRepository`
  Istanza della porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (paragrafo 4.4.18.1);
- `threadChangesTrackerRepository: ThreadChangesTrackerRepository`
  Porta che permette di recuperare dalla persistenza una specifica entità che implementi *ThreadChangesTracker* (paragrafo 4.4.18.14).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.12.4 ChangesThreadMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il Builder pattern per costruire l'oggetto *ChangesThreadMethodResponsePort* (vedi paragrafo 4.4.12.7).

Metodi:

- `accountId(accountId: String): ChangesThreadMethodResponseBuilderPort`
  Id dell'account di cui si stanno ritornando le modifiche;
- `oldState(oldState: String): ChangesThreadMethodResponseBuilderPort`
  Stato del client prima di applicare tutti i cambiamenti ritornati, ovvero lo stato inviato durante la chiamata;
- `newState(newState: String): ChangesThreadMethodResponseBuilderPort`
  Nuovo stato del client dopo aver applicato tutti i cambiamenti presenti nella risposta alla chiamata;
- `hasMoreChanges(hasMoreChanges: Boolean): ChangesThreadMethodResponseBuilderPort`
  Notifica il client che i cambiamenti ritornati non sono tutti, ma solo una parte;
- `created(ids: String[0..*]): ChangesThreadMethodResponseBuilderPort`
  Id dei thread creati dall'ultimo stato del client fino a quello attuale del server;
- `updated(ids: String[0..*]): ChangesThreadMethodResponseBuilderPort`
  Id dei thread modificati dall'ultimo stato del client fino a quello attuale del server;
- `destroyed(ids: String[0..*]): ChangesThreadMethodResponseBuilderPort`
  Id dei thread distrutti dall'ultimo stato del client fino a quello attuale del server;
- `build(): ChangesThreadMethodResponsePort`
  Crea l'oggetto *ChangesThreadMethodResponsePort* (vedi paragrafo 4.4.12.7);
- `reset(): GetThreadMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.12.5 ChangesThreadMethodResponseBuilderAdapter

Implementa l'interfaccia *ChangesThreadMethodResponseBuilderPort* (paragrafo 4.4.12.4).

Attributi:

- `changesThreadMethodResponseBuilder: ChangesMethodResponseBuilder`
  Builder della libreria JMAP.

4.4.12.6 ChangesThreadMethodResponseBuilderAdapter

Implementa l'interfaccia *ChangesThreadMethodResponseBuilderPort* (paragrafo 4.4.12.4).

Attributi:

- `changesThreadMethodResponseBuilder: ChangesThreadMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Thread/Changes.

4.4.12.7 ChangesThreadMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Thread/Changes.

4.4.12.8 ChangesThreadMethodResponseAdapter

Implementa l'interfaccia *ChangesThreadMethodResponsePort* (paragrafo 4.4.12.7).

Attributi:

- `changesThreadMethodResponse: ChangesThreadMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Thread/Changes.

Metodi:

- `adaptee(): ChangesThreadMethodResponse`
  Ritorna l'adaptee.

4.4.12.9 ChangesThreadMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Thread/Changes. Estende l'interfaccia di base *ChangesMethodCallPort* (paragrafo 4.4.18.29).

4.4.12.10 ChangesThreadMethodCallAdapter

Implementa l'interfaccia *ChangesThreadMethodCallPort* (paragrafo 4.4.12.9).

Attributi:

- `changesThreadMethodCall: ChangesThreadMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Thread/Changes.

Metodi:

- `adaptee(): ChangesThreadMethodCall`
  Ritorna l'adaptee.

4.4.13 SetEmailMethodCall

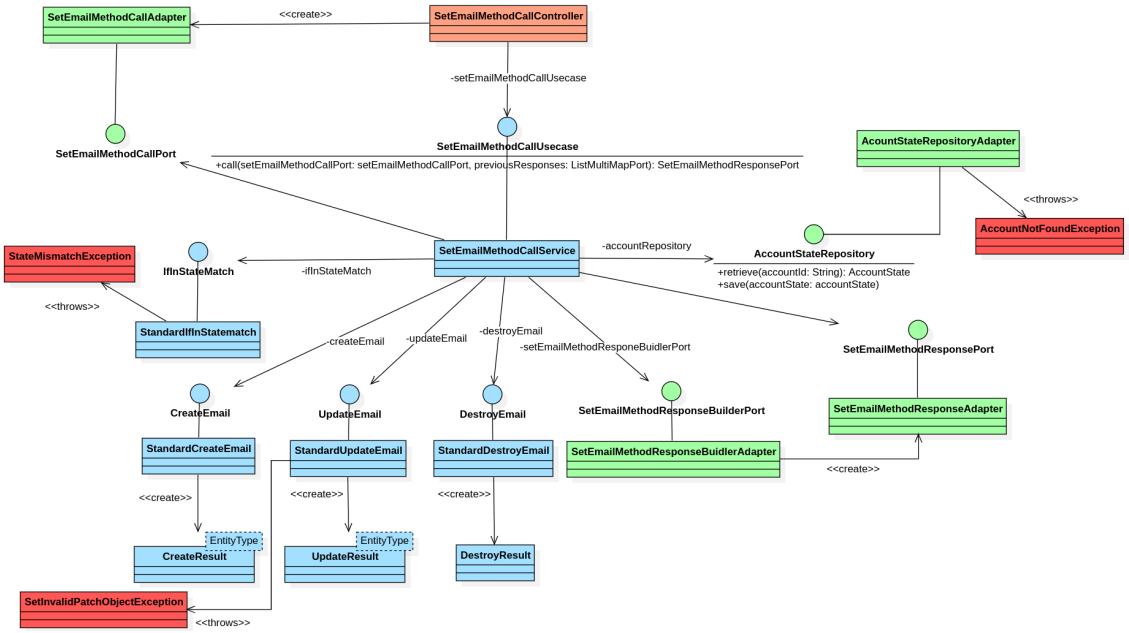

Figura 15: Diagramma delle classi che implementano la Email/Set.

4.4.13.1 Riferimenti

- Per **SetEmailMethodCallController** vedi la sezione 4.4.2.16;
- per **StandardIfInStateMatch** vedi la sezione 4.4.18.24;
- per **IfInStateMatch** vedi la sezione 4.4.18.23;
- per **CreatedResult<EntityType>** vedi la sezione 4.4.18.25;

- per **UpdatedResult<EntityType>** vedi la sezione 4.4.18.26;
- per **DestroyedResult** vedi la sezione 4.4.18.27;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **EmailRepository** vedi la sezione 4.4.19.3;
- per **EmailRepositoryAdapter** vedi la sezione 4.4.19.4;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.13.2 SetEmailMethodCallUsecase

Dichiara il contratto che deve avere il service che elabora la richiesta Email/Set e genera la corrispettiva risposta dove vengono riportati tutte le email create, aggiornate e distrutte.

Metodi:

- `call(setEmailMethodCallPort: SetEmailMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>) SetEmailMethodResponsePort`
  Riceve una chiamata JMAP Email/Set e genera la corrispettiva risposta.

4.4.13.3 SetEmailMethodCallService

Implementa l'interfaccia *SetEmailMethodCallUsecase* (paragrafo 4.4.13.2).

Attributi:

- `accountStateRepository: AccountStateRepository`
  Repository per recuperare l'entità *AccountState* (paragrafo 4.4.18.1) dalla persistenza;
- `ifInStateMatch: IfInStateMatch`
  Come deve essere fatto il controllo dello stato corrente del server rispetto a quello della chiamata;

- `createEmail: CreateEmail`
  Come vengono create le email;
- `updateEmail: UpdateEmail`
  Come vengono aggiornate le email;
- `destroyEmail: DestroyEmail`
  Come vengono distrutte le email;
- `setEmailMethodResponseBuilderPort: SetEmailMethodResponseBuilderPort`
  Pattern builder per costruire la risposta alla chiamata JMAP Email/Set.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.13.4 CreateEmail

Interfaccia che dichiara il contratto che devono avere le classi che implementano la creazione dell'email.

Metodi:

- `create(setEmailMethodCallPort: SetEmailMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): CreatedResult<EmailPort>`
  Crea le email presenti nella *SetEmailMethodCallPort* (paragrafo 4.4.13.14) e ritorna come risultato *CreatedResult<EmailPort>* (paragrafo 4.4.18.25).

4.4.13.5 StandardCreateEmail

Implementa l'interfaccia *CreateEmail* (paragrafo 4.4.13.4) come da Standard JMAP.

Attributi:

- `emailRepository: EmailRepository`
  Permette di recuperare l'entità *EmailPort* (paragrafo 4.4.18.2) dalla persistenza;

- `emailBuilderPort: emailBuilderPort`
  Permette di costruire l'entità *EmailPort* (paragrafo 4.4.18.2) ;
- `threadRepository: threadRepository`
  Permette di recuperare l'entità *ThreadPort* (paragrafo 4.4.18.19) dalla persistenza;
- `emailChangesTrackerRepository`
  Permette di recuperare l'entità *EmailChangesTracker* (paragrafo 4.4.18.8) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `threadChangesTrackerRepository`
  Permette di recuperare l'entità *ThreadChangesTracker* (paragrafo 4.4.18.14) dalla persistenza;
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;
- `setErrorEnumPort: SetErrorEnumPort`
  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20);
- `creationIdResolverPort: CreationIdResolverPort`
  Permette di risolvere ids impostati dal client, con gli ids impostati dal server durante l'esecuzione dei comandi /Set nelle chiamate precedenti della richiesta.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.13.6 UpdateEmail

Interfaccia che dichiara il contratto che devono avere le classi che implementano l'update delle email.

Metodi:

- `create(setEmailMethodCallPort: SetEmailMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): CreatedResult<EmailPort>`
  Aggiorna le email presenti nella *SetEmailMethodCallPort* (paragrafo 4.4.13.14) e ritorna come risultato *UpdatedResult<EmailPort>* (paragrafo 4.4.18.26);
- `create(setEmailSubmissionMethodCallPort: SetEmailSubmissionMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>, successSubmissionEmailIdResolve: Map<String,String>): CreatedResult<EmailPort>`
  Aggiorna l'e-mail presenti nella *SetEmailSubmissionMethodCallPort* (paragrafo 4.4.14.8) e ritorna come risultato *UpdatedResult<EmailPort>* (paragrafo 4.4.18.26).

4.4.13.7 StandardUpdateEmail

Implementa l'interfaccia *UpdateEmail* (paragrafo 4.4.13.6) come da standard JMAP.

Attributi:

- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;
- `emailChangesTrackerRepository`
  Permette di recuperare l'entità *EmailChangesTracker* (paragrafo 4.4.18.8) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `creationIdResolverPort: CreationIdResolverPort`
  Permette di risolvere ids impostati dal client, con gli ids impostati dal server durante l'esecuzione dei comandi /Set nelle chiamate precedenti della richiesta.

Metodi:

- `applayPatches(accountId: String, emailIdToPatch: String, patchObjects: Map<String, Object>, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): EmailPort`
  Applica la patch ad una singola email e poi ritorna eventuali ulteriori modifiche che la patch può aver causato.

4.4.13.8 DestroyEmail

Interfaccia che dichiara il contratto che devono avere le classi che implementano la distruzione delle email.

Metodi:

- `destroy(setEmailMethodCallPort: SetEmailMethodCallPort): DestroyedResult`
  Distrugge le email presenti nella *SetEmailMethodCallPort* (paragrafo 4.4.13.14) e ritorna come risultato *DestroyedResult<EmailPort>* (paragrafo 4.4.18.27).

4.4.13.9 StandardDestroyEmail

Implementa l'interfaccia *DestroyEmail* (paragrafo 4.4.13.8) come da Standard JMAP.

Attributi:

- `emailRepository: EmailRepository`
  Permette di recuperare l'entità *EmailPort* (paragrafo 4.4.18.2) dalla persistenza;
- `emailChangesTrackerRepository`
  Permette di recuperare l'entità *EmailChangesTracker* (paragrafo 4.4.18.8) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `threadChangesTrackerRepository`
  Permette di recuperare l'entità *ThreadChangesTracker* (paragrafo 4.4.18.14) dalla persistenza;

- `setErrorEnumPort: SetErrorEnumPort`
  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20);
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza.

4.4.13.10 SetEmailMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il pattern builder per costruire l'oggetto *SetEmailMethodResponsePort* (paragrafo 4.4.13.12) .

Metodi:

- `accountId(accountId: String): SetEmailMethodResponseBuilderPort`
  Imposta l'account id della risposta;
- `oldState(oldState: String): SetEmailMethodResponseBuilderPort`
  Imposta lo stato da cui è partito il /Set a fare modifiche;
- `newState(newState: String): SetEmailMethodResponseBuilderPort`
  Imposta lo stato raggiunto dal server applicando le modifiche richieste;
- `created(created: Map<String,EmailPort>): SetEmailMethodResponseBuilderPort`
  Imposta il risultato delle email create;
- `updated(updated: Map<String,EmailPort>): SetEmailMethodResponseBuilderPort`
  Imposta il risultato delle email aggiornate;
- `destroyed(destroyed: String[0..*]): SetEmailMethodResponseBuilderPort`
  Imposta il risultato delle email distrutte;
- `notCreated(notCreated: Map<String,SetErrorPort>): SetEmailMethodResponseBuilderPort`
  Imposta il risultato delle email non create con i relativi errori;
- `notUpdated(notUpdated: Map<String,SetErrorPort>): SetEmailMethodResponseBuilderPort`

Imposta il risultato delle email non aggiornate con i relativi errori;

- `notDestroyed(notDestroyed: Map<String,SetErrorPort>): SetEmailMethodResponseBuilderPort`
  Imposta il risultato delle email non distrutte con i relativi errori;
- `build(): SetEmailMethodResponsePort`
  Crea la risposta *SetEmailMethodResponsePort* (paragrafo 4.4.13.12) ;
- `reset(): SetEmailMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.13.11 SetEmailMethodResponseBuilderAdapter

Implementa l'interfaccia *SetEmailMethodResponseBuilderPort* (paragrafo 4.4.13.10) .

Attributi:

- `setEmailMethodResponseBuilder: SetEmailMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Email/Set.

4.4.13.12 SetEmailMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Email/Set.

4.4.13.13 SetEmailMethodResponseAdapter

Implementa l'interfaccia *SetEmailMethodResponsePort* (paragrafo 4.4.13.12) .

Attributi:

- `setEmailMethodResponse: SetEmailMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Email/Set.

Metodi:

- `adaptee(): SetEmailMethodResponse`
  Ritorna l'adaptee.

4.4.13.14 SetEmailMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Email/Query.

Metodi:

- `accountId(): String`
  Id account target della chiamata;
- `ifInState(): String`
  Stato del client al momento della creazione chiamata;
- `getCreate(): Map<String, EmailPort>`
  Email da creare;
- `getUpdate(): Map<String, Map<String,Object»`
  Email da aggiornare;
- `getDestroy(): String[0..*]`
  Email da distruggere.

4.4.13.15 SetEmailMethodCallAdapter

Implementa l'interfaccia *SetEmailMethodCallPort* (paragrafo 4.4.13.14) .

Attributi:

- `setEmailMethodCall: SetEmailMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Email/Set.

Metodi:

- `adaptee(): SetEmailMethodCall`
  Ritorna l'adaptee.

4.4.14 SetEmailSubmissionMethodCall

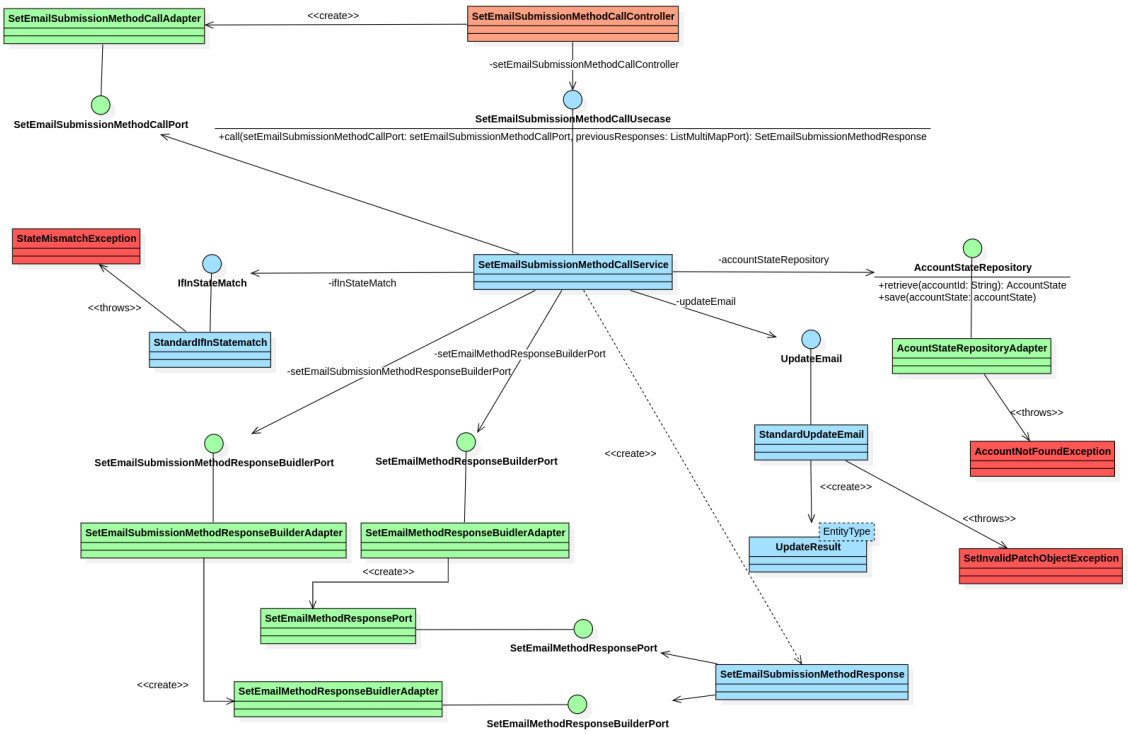

Figura 16: Diagramma delle classi che implementano la EmailSubmission/Set.

4.4.14.1 Riferimenti

- Per **SetEmailMethodCallController** vedi la sezione 4.4.2.16;
- per **IfInStateMatch** vedi la sezione 4.4.18.23;
- per **StandardIfInStateMatch** vedi la sezione 4.4.18.24;
- per **CreatedResult<EntityType>** vedi la sezione 4.4.18.25;
- per **UpdateEmail** vedi la sezione 4.4.13.6;
- per **StandardUpdateEmail** vedi la sezione 4.4.13.7;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **SetEmailMethodResponseBuilderPort** vedi la sezione 4.4.13.10;
- per **SetEmailMethodResponseBuilderAdapter** vedi la sezione 4.4.13.11;

- per **SetEmailMethodResponsePort** vedi la sezione 4.4.13.12;
- per **SetEmailMethodResponseAdapter** vedi la sezione 4.4.13.13;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.14.2 SetEmailSubmissionMethodCallUsecase

Dichiara il contratto che deve avere il service che elabora la richiesta EmailSubmission/Set e genera la corrispettiva risposta.

Metodi:

- `call(setEmailSubmissionMethodCallPort: SetEmailSubmissionMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>) SetIdentityMethodResponsePort`
  Riceve una chiamata JMAP EmailSubmission/Set e genera la corrispettiva risposta.

4.4.14.3 SetEmailSubmissionMethodCallService

Implementa l'interfaccia *SetIdentityMethodCallUsecase* (paragrafo 4.4.15.2).

Attributi:

- `accountStateRepository: AccountStateRepository`
  Repository per recuperare l'entità *AccountState* (paragrafo 4.4.18.1) dalla persistenza;
- `ifInStateMatch: IfInStateMatch`
  Come deve essere fatto il controllo dello stato corrente del server rispetto a quello della chiamata;
- `updateEmail: UpdateEmail`
  Come vengono aggiornate le email;
- `setEmailSubmissionMethodResponseBuilderPort: SetEmailSubmissionMethodResponseBuilderPort`
  Pattern builder per costruire la risposta alla chiamata JMAP EmailSubmission/Set;

- `setEmailMethodResponseBuilderPort: SetEmailMethodResponseBuilderPort`
  Pattern builder per costruire la risposta alla chiamata JMAP Email/Set.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.14.4 SetEmailSubmissionMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il pattern builder per costruire l'oggetto *SetEmailSubmissionMethodResponsePort* (paragrafo 4.4.14.6) .

Metodi:

- `accountId(accountId: String): SetEmailSubmissionMethodResponseBuilderPort`
  Imposta l'account della risposta;
- `oldState(oldState: String): SetEmailSubmissionMethodResponseBuilderPort`
  Imposta lo stato da cui è partito il /Set a fare modifiche;
- `newState(newState: String): SetEmailSubmissionMethodResponseBuilderPort`
  Imposta lo stato raggiunto dal server applicando le modifiche richieste;
- `created(created: Map<String,MailboxPort>): SetEmailSubmissionMethodResponseBuilderPort`
  Imposta il risultato delle mailbox create;
- `build(): SetEmailSubmissionMethodResponsePort`
  Crea la risposta *SetEmailSubmissionMethodResponsePort* (paragrafo 4.4.14.6) ;
- `reset(): SetEmailSubmissionMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.14.5 SetEmailSubmissionmethodResponseBuilderAdapter

Implementa l'interfaccia *SetEmailSubmissionMethodResponseBuilderPort* (paragrafo 4.4.14.4) .

Attributi:

- `setEmailSubmissionMethodResponseBuilder: SetEmailSubmissionMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP EmailSubmission/Set.

4.4.14.6 SetEmailSubmissionMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata EmailSubmission/Set.

4.4.14.7 SetEmailSubmissionMethodResponseAdapter

Implementa l'interfaccia *SetEmailSubmissionMethodResponsePort* (paragrafo 4.4.14.6) .

Attributi:

- `setEmailSubmissionMethodResponse: SetEmailSubmissionMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP EmailSubmission/Set.

Metodi:

- `adaptee(): SetEmailSubmissionMethodResponse`
  Ritorna l'adaptee.

4.4.14.8 SetEmailSubmissionMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP EmailSubmission/Set.

Metodi:

- `accountId(): String`
  Ritorna l'accountId target della chiamata;
- `getIfInState(): String`
  Ritorna lo stato in cui il client aveva fatto la chiamata;
- `getOnSuccessUpdateEmail(): Map<String, Map<String, Object»`
  Ottiene gli update per le email quando la submission va a buon fine;
- `getCreate(): Map<String, Map<String, Object»`
  Ottiene le submission da creare.

4.4.14.9 SetEmailSubmissionMethodCallAdapter

Implementa l'interfaccia *SetEmailSubmissionMethodCallPort* (paragrafo 4.4.14.8) .

Attributi:

- `setEmailSubmissionMethodCall: SetEmailSubmissionMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP EmailSubmission/Set.

Metodi:

- `adaptee(): SetEmailSubmissionMethodCall`
  Ritorna l'adaptee.

4.4.14.10 SetEmailSubmissionMethodResponse

Record che definisce l'insieme di risposte che genera la chiamata JMAP EmailSubmission/Set.

Attributi/Metodi (record):

- `setEmailSubmissionMethodResponsePort: SetEmailSubmissionMethodResponsePort`
  Risposta EmailSubmission/Set;

- `setEmailMethodResponsePort: SetEmailMethodResponsePort`

  Risposta Email/Set.

4.4.15 SetIdentityMethodCall

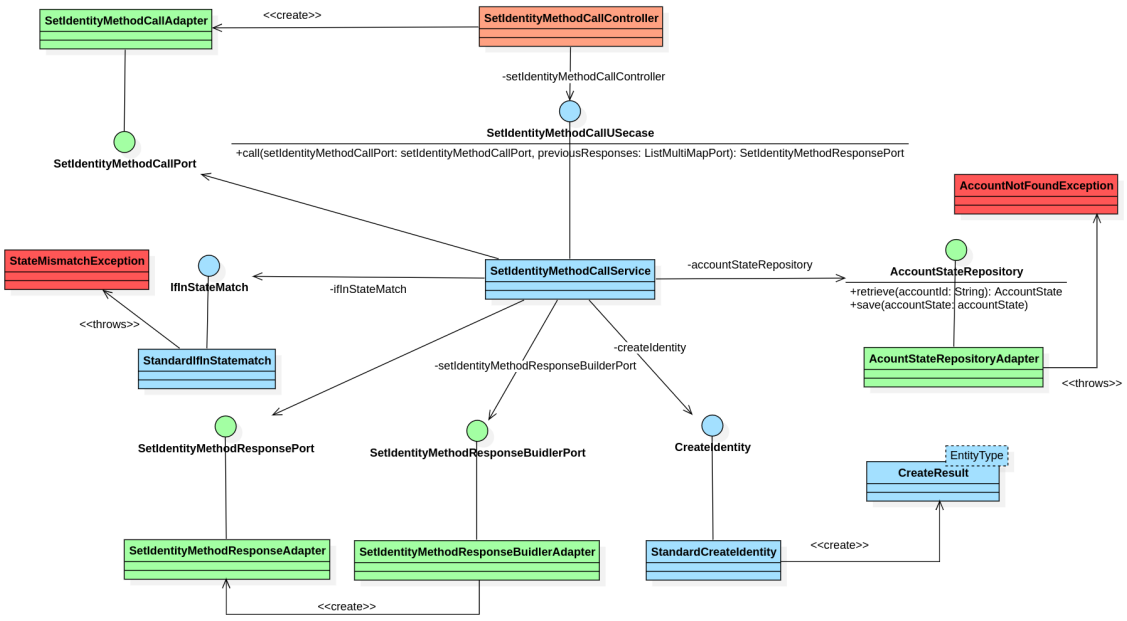

Figura 17: Diagramma delle classi che implementano la Identity/Set.

4.4.15.1 Riferimenti

- Per **SetEmailMethodCallController** vedi la sezione 4.4.2.16;
- per **IfInStateMatch** vedi la sezione 4.4.18.23;
- per **StandardIfInStateMatch** vedi la sezione 4.4.18.24;
- per **UpdatedResult<EntityType>** vedi la sezione 4.4.18.26;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.15.2 SetIdentityMethodCallUsecase

Dichiara il contratto che deve avere il service che elabora la richiesta Identity/Set e genera la corrispettiva risposta dove vengono riportati tutte le identities create, aggiornate e distrutte.

Metodi:

- `call(setIdentityMethodCallPort: SetIdentityMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>) SetIdentityMethodResponsePort`
  Riceve una chiamata JMAP Identity/Set e genera la corrispettiva risposta.

4.4.15.3 SetIdentityMethodCallService

Implementa l'interfaccia *SetIdentityMethodCallUsecase* (paragrafo 4.4.15.2).

Attributi:

- `accountStateRepository: AccountStateRepository`
  Repository per recuperare l'entità *AccountState* (paragrafo 4.4.18.1) dalla persistenza;
- `ifInStateMatch: IfInStateMatch`
  Come deve essere fatto il controllo dello stato corrente del server rispetto a quello della chiamata;
- `createIdentity: CreateIdentity`
  Come vengono create le identities;
- `updateIdentity: UpdateIdentity`
  Come vengono aggiornate le identities;
- `destroyIdentity: DestroyIdentity`
  Come vengono distrutte le identities;
- `setIdentityMethodResponseBuilderPort: SetIdentityMethodResponseBuilderPort`
  Pattern builder per costruire la risposta alla chiamata JMAP Identity/Set.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.15.4 CreateIdentity

Interfaccia che dichiara il contratto che devono avere le classi che implementano la creazione delle identities.

Metodi:

- `create(setIdentityMethodCallPort: SetIdentityMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): CreatedResult<IdentityPort>`
  Crea le identities presenti nella *SetIdentityMethodCallPort* (paragrafo 4.4.15.10) e ritorna come risultato *CreatedResult<IdentityPort>* (paragrafo 4.4.18.25).

4.4.15.5 StandardCreateIdentity

Implementa l'interfaccia *CreateIdentity* (paragrafo 4.4.15.4) come da Standard JMAP.

Attributi:

- `identityRepository: IdentityRepository`
  Permette di recuperare l'entità *IdentityPort* (paragrafo 4.4.18.18) dalla persistenza;
- `identityChangesTrackerRepository`
  Permette di recuperare l'entità *IdentityChangesTracker* (paragrafo 4.4.18.12) dalla persistenza;
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;
- `setErrorEnumPort: SetErrorEnumPort`
  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.15.6 SetIdentityMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il pattern builder per costruire l'oggetto *SetIdentityMethodResponsePort* (paragrafo 4.4.15.8) .

Metodi:

- `accountId(accountId: String): SetIdentityMethodResponseBuilderPort`
  Imposta l'account della risposta;
- `oldState(oldState: String): SetIdentityMethodResponseBuilderPort`
  Imposta lo stato da cui è pratito il /Set a fare modifiche;
- `newState(newState: String): SetIdentityMethodResponseBuilderPort`
  Imposta lo stato raggiunto dal server applicando le modifiche richieste;
- `created(created: Map<String,IdentityPort>): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities create;
- `updated(updated: Map<String,IdentityPort>): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities aggiornate;
- `destroyed(destroyed: String[0..*]): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities distrutte;
- `notCreated(notCreated: Map<String,SetErrorPort>): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities non create con i relativi errori;
- `notUpdated(notUpdated: Map<String,SetErrorPort>): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities non aggiornate con i relativi errori;

- `notDestroyed(notDestroyed: Map<String,SetErrorPort>): SetIdentityMethodResponseBuilderPort`
  Imposta il risultato delle identities non distrutte con i relativi errori;
- `build(): SetIdentityMethodResponsePort`
  Crea la risposta *SetIdentityMethodResponsePort* (paragrafo 4.4.15.8) ;
- `reset(): SetIdentityMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.15.7 SetIdentityMethodResponseBuilderAdapter

Implementa l'interfaccia *SetIdentityMethodResponseBuilderPort* (paragrafo 4.4.15.6) .

Attributi:

- `setIdentityMethodResponseBuilder: SetIdentityMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Identity/Set.

4.4.15.8 SetIdentityMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Identity/Set.

4.4.15.9 SetIdentityMethodResponseAdapter

Implementa l'interfaccia *SetIdentityMethodResponsePort* (paragrafo 4.4.15.8) .

Attributi:

- `setIdentityMethodResponse: SetIdentityMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Identity/Set.

Metodi:

- `adaptee(): SetIdentityMethodResponse`
  Ritorna l'adaptee.

4.4.15.10 SetIdentityMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Identity/Set.

Metodi:

- `accountId(): String`
  Id account target della chiamata;
- `ifInState(): String`
  Stato del client al momento della creazione chiamata;
- `getCreate(): Map<String, IdentityPort>`
  Identities da creare;
- `getUpdate(): Map<String, Map<String,Object»`
  Identities da aggiornare;
- `getDestroy(): String[0..*]`
  Identities da distruggere.

4.4.15.11 SetIdentityMethodCallAdapter

Implementa l'interfaccia *SetIdentityMethodCallPort* (paragrafo 4.4.15.10) .

Attributi:

- `setIdentityMethodCall: SetIdentityMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Identity/Set.

Metodi:

- `adaptee(): SetIdentityMethodCall`
  Ritorna l'adaptee.

4.4.16 SetMailboxMethodCall

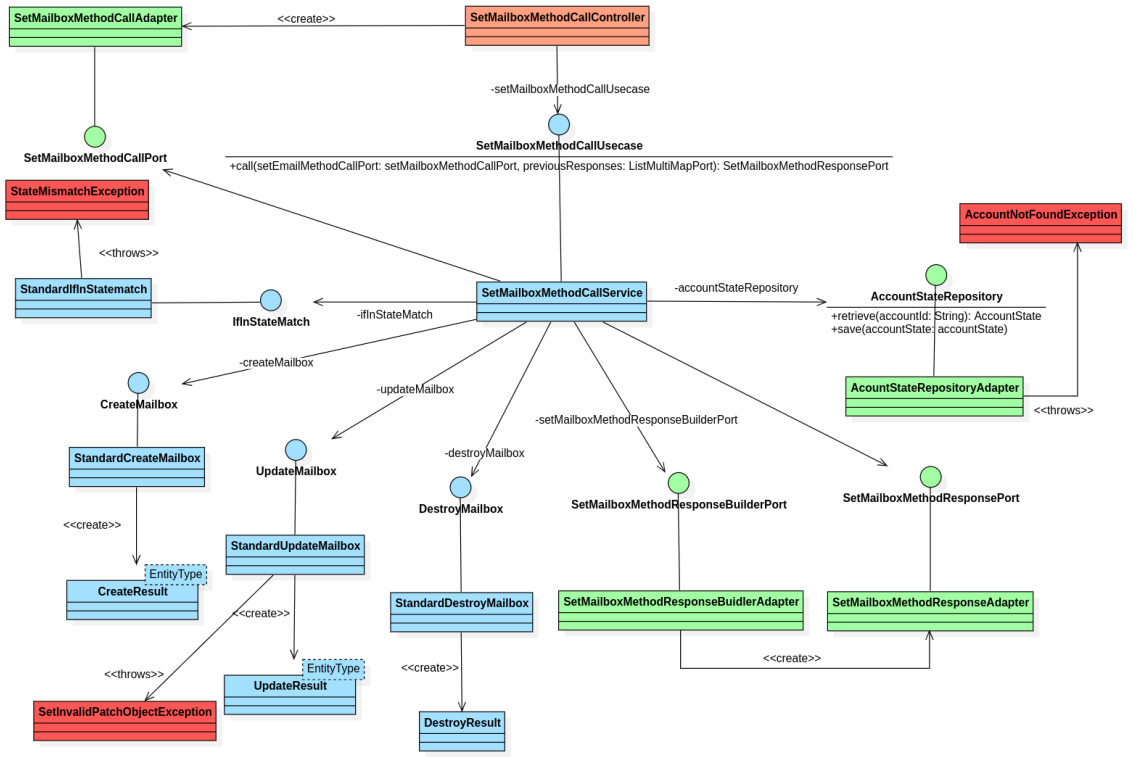

Figura 18: Diagramma delle classi che implementano la Mailbox/Set.

4.4.16.1 Riferimenti

- Per **SetMailboxMethodCallController** vedi la sezione 4.4.2.27;
- per **IfInStateMatch** vedi la sezione 4.4.18.23;
- per **StandardIfInStateMatch** vedi la sezione 4.4.18.24;
- per **CreatedResult<EntityType>** vedi la sezione 4.4.18.25;
- per **UpdatedResult<EntityType>** vedi la sezione 4.4.18.26;
- per **DestroyedResult** vedi la sezione 4.4.18.27;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **MailboxRepository** vedi la sezione 4.4.19.5;
- per **MailboxRepositoryAdapter** vedi la sezione 4.4.19.6;

- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.16.2 SetMailboxMethodCallUsecase

Dichiara il contratto che deve avere il service che elabora la richiesta Mailbox/Set e genera la corrispettiva risposta dove vengono riportati tutte le mailbox create, aggiornate e distrutte.

Metodi:

- `call(setMailboxMethodCallPort: SetMailboxMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>) SetMailboxMethodResponsePort`
  Riceve una chiamata JMAP Mailbox/Set e genera la corrispettiva risposta.

4.4.16.3 SetMailboxMethodCallService

Implementa l'interfaccia *SetMailboxMethodCallUsecase* (paragrafo 4.4.16.2).

Attributi:

- `accountStateRepository: AccountStateRepository`
  Repository per recuperare l'entità *AccountState* (paragrafo 4.4.18.1) dalla persistenza;
- `ifInStateMatch: IfInStateMatch`
  Come deve essere fatto il controllo dello stato corrente del server rispetto a quello della chiamata;
- `createMailbox: CreateMailbox`
  Come vengono create le mailbox;
- `updateMailbox: UpdateMailbox`
  Come vengono aggiornate le mailbox;
- `destroyMailbox: DestroyMailbox`
  Come vengono distrutte le mailbox;
- `setMailboxMethodResponseBuilderPort: SetMailboxMethodResponseBuilderPort`

Pattern builder per costruire la risposta alla chiamata JMAP Mailbox/Set.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.16.4 CreateMailbox

Interfaccia che dichiara il contratto che devono avere le classi che implementano la creazione delle mailbox.

Metodi:

- `create(setMailboxMethodCallPort: SetMailboxMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): CreatedResult<MailboxPort>`
  Crea le mailbox presenti nella *SetMailboxMethodCallPort* (paragrafo 4.4.16.14) e ritorna come risultato *CreatedResult<MailboxPort>* (paragrafo 4.4.18.25).

4.4.16.5 StandardCreateMailbox

Implementa l'interfaccia *CreateMailbox* (paragrafo 4.4.16.4) come da Standard JMAP.

Attributi:

- `mailboxRepository: MailboxRepository`
  Permette di recuperare l'entità *MailboxPort* (paragrafo 4.4.18.6) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;
- `setErrorEnumPort: SetErrorEnumPort`
  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20).

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.16.6 UpdateMailbox

Interfaccia che dichiara il contratto che devono avere le classi che implementano l'aggiornamento delle mailbox.

Metodi:

- `update(setMailboxMethodCallPort: SetMailboxMethodCallPort, previousResponses: ListMultimapPort<String,ResponseInvocationPort>): UpdatedResult<MailboxPort>`
  Aggiorna le mailbox presenti nella *SetMailboxMethodCallPort* (paragrafo 4.4.16.14) e ritorna come risultato *UpdatedResult<MailboxPort>* (paragrafo 4.4.18.26).

4.4.16.7 StandardUpdateMailbox

Implementa l'interfaccia *UpdateMailbox* (paragrafo 4.4.16.6) come da Standard JMAP.

Attributi:

- `mailboxRepository: MailboxRepository`
  Permette di recuperare l'entità *MailboxPort* (paragrafo 4.4.18.6) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;
- `setErrorEnumPort: SetErrorEnumPort`
  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20);

- `creationIdResolverPort: CreationIdResolverPort`
  Permette di risolvere gli ids impostati dal client, con gli ids impostati dal server durante l'esecuzione dei comandi /Set nelle chiamate precedenti della richiesta.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.16.8 DestroyMailbox

Interfaccia che dichiara il contratto che devono avere le classi che implementano la distruzione delle mailbox.

Metodi:

- `destroy(setMailboxMethodCallPort: SetMailboxMethodCallPort): DestroyedResult`
  Distrugge le mailbox presenti nella *SetMailboxMethodCallPort* (paragrafo 4.4.16.14) e ritorna come risultato *DestroyedResult<MailboxPort>* (paragrafo 4.4.18.27).

4.4.16.9 StandardDestroyMailbox

Implementa l'interfaccia *DestroyMailbox* (paragrafo 4.4.16.8) come da Standard JMAP.

Attributi:

- `mailboxRepository: MailboxRepository`
  Permette di recuperare l'entità *MailboxPort* (paragrafo 4.4.18.6) dalla persistenza;
- `mailboxChangesTrackerRepository`
  Permette di recuperare l'entità *MailboxChangesTracker* (paragrafo 4.4.18.10) dalla persistenza;
- `accountStateRepository: AccountStateRepository`
  Permette di recuperare l'entità *AccountState* 4.4.18.1 dalla persistenza;

- `setErrorEnumPort: SetErrorEnumPort`

  Permette di creare *SetErrorPort* (paragrafo 4.4.18.20).

4.4.16.10 SetMailboxMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il pattern builder per costruire l'oggetto *SetMailboxMethodResponsePort* (paragrafo 4.4.16.12) .

Metodi:

- `accountId(accountId: String): SetMailboxMethodResponseBuilderPort`

  Imposta l'account della risposta;

- `oldState(oldState: String): SetMailboxMethodResponseBuilderPort`

  Imposta lo stato da cui è partito il /Set a fare modifiche;

- `newState(newState: String): SetMailboxMethodResponseBuilderPort`

  Imposta lo stato raggiunto dal server applicando le modifiche richieste;

- `created(created: Map<String,MailboxPort>): SetMailboxMethodResponseBuilderPort`

  Imposta il risultato delle mailbox create;

- `updated(updated: Map<String,MailboxPort>): SetMailboxMethodResponseBuilderPort`

  Imposta il risultato delle mailbox aggiornate;

- `destroyed(destroyed: String[0..*]): SetMailboxMethodResponseBuilderPort`

  Imposta il risultato delle mailbox distrutte;

- `notCreated(notCreated: Map<String,SetErrorPort>): SetMailboxMethodResponseBuilderPort`

  Imposta il risultato delle mailbox non create con i relativi errori;

- `notUpdated(notUpdated: Map<String,SetErrorPort>): SetMailboxMethodResponseBuilderPort`

  Imposta il risultato delle mailbox non aggiornate con i relativi errori;

- `notDestroyed(notDestroyed: Map<String,SetErrorPort>): SetMailboxMethodResponseBuilderPort`
  Imposta il risultato delle mailbox non distrutte con i relativi errori;
- `build(): SetMailboxMethodResponsePort`
  Crea la risposta *SetMailboxMethodResponsePort* (paragrafo 4.4.16.12) ;
- `reset(): SetMailboxMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.16.11 SetMailboxMethodResponseBuilderAdapter

Implementa l'interfaccia *SetMailboxMethodResponseBuilderPort* (paragrafo 4.4.16.10) .

Attributi:

- `setMailboxMethodResponseBuilder: SetMailboxMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Mailbox/Set.

4.4.16.12 SetMailboxMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Mailbox/Set.

4.4.16.13 SetMailboxMethodResponseAdapter

Implementa l'interfaccia *SetMailboxMethodResponsePort* (paragrafo 4.4.16.12) .

Attributi:

- `setMailboxMethodResponse: SetMailboxMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Mailbox/Set.

Metodi:

- `adaptee(): SetMailboxMethodResponse`
  Ritorna l'adaptee.

4.4.16.14 SetMailboxMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Mailbox/Set.

Metodi:

- `accountId(): String`
  Id account target della chiamata;
- `ifInState(): String`
  Stato del client al momento della creazione chiamata;
- `getCreate(): Map<String, MailboxPort>`
  Mailbox da creare;
- `getUpdate(): Map<String, Map<String,Object»`
  Mailbox da aggiornare;
- `getDestroy(): String[0..*]`
  Mailbox da distruggere.

4.4.16.15 SetMailboxMethodCallAdapter

Implementa l'interfaccia *SetMailboxMethodCallPort* (paragrafo 4.4.16.14) .

Attributi:

- `setMailboxMethodCall: SetMailboxMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Mailbox/Set.

Metodi:

- `adaptee(): SetMailboxMethodCall`
  Ritorna l'adaptee.

4.4.17 QueryEmailMethodCall

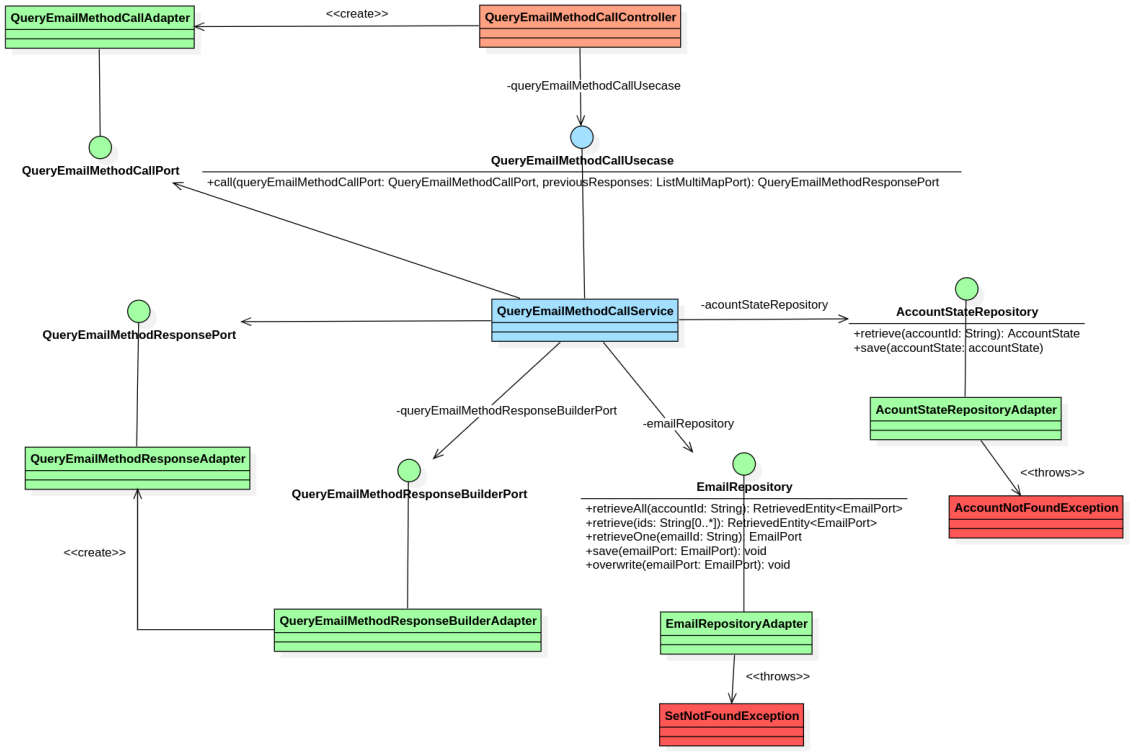

Figura 19: Diagramma delle classi che implementano la Email/Query.

4.4.17.1 Riferimenti

- Per **QueryEmailMethodCallController** vedi la sezione 4.4.2.24;
- per **AccountStateRepository** vedi la sezione 4.4.19.1;
- per **AccountStateRepositoryAdapter** vedi la sezione 4.4.19.2;
- per **EmailRepository** vedi la sezione 4.4.19.3;
- per **EmailRepositoryAdapter** vedi la sezione 4.4.19.4;
- per la definizione delle **eccezioni** vedi la paragrafo 4.4.20.

4.4.17.2 QueryEmailMethodCallUsecase

Dichiara il contratto che deve avere il service che elabora la richiesta Email/Query e genera la corrispettiva risposta dove vengono riportate tutte le email recuperate dalla query indicata nella chiamata.

Metodi:

- `call(queryEmailMethodCallPort: QueryEmailMethodCallPort, ListMultimapPort<String,ResponseInvocationPort> previousResponses): QueryEmailMethodResponsePort`
  Elabora la chiamata ricevuta e ritorna il risultato della query.

4.4.17.3 QueryEmailMethodCallService

Implementa l'interfaccia *QueryEmailMethodCallUsecase* (sezione 4.4.17.2).

Attributi:

- `queryEmailMethodResponseBuilderPort: queryEmailMethodResponseBuilderPort`
  Istanza di una porta che implementi il pattern builder per l'entità *QueryEmailMethodResponsePort* (sezione 4.4.17.6);
- `accountStateRepository: AccountStateRepository`
  Istanza della porta che permetta di recuperare dalla persistenza l'oggetto *AccountState* (sezione 4.4.18.1);
- `emailRepository: EmailRepository`
  Istanza della porta repository per recuperare l'email dalla persistenza.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.17.4 QueryEmailMethodResponseBuilderPort

Dichiara il contratto che deve avere la classe che implementi il pattern builder per costruire l'oggetto *QueryEmailMethodResponsePort* (vedi sezione 4.4.17.6).

Metodi:

- `accountId(accountId: String): QueryEmailMethodResponseBuilderPort`
  Imposta l'account della risposta;

- `queryState(queryState: String): QueryEmailMethodResponseBuilderPort`
  Lo stato della query;
- `canCalculateChanges(canCalculateChanges: Boolean): QueryEmailMethodResponseBuilderPort`
  Indica se possono essere recuperati i cambiamenti di tutti gli elementi ritornati;
- `position(position: Long): QueryEmailMethodResponseBuilderPort`
  Zero-base indice del primo id se l'intera lista è stata ritornata;
- `ids(ids: String[0..*]): QueryEmailMethodResponseBuilderPort`
  Ids delle email che sono state recuperate dalla query;
- `total(total: Long): QueryEmailMethodResponseBuilderPort`
  Email totali nella risposta;
- `limit(limit: Long): QueryEmailMethodResponseBuilderPort`
  Limite di email nella risposta;
- `build(): QueryEmailMethodResponsePort`
  Crea l'oggetto *QueryEmailMethodResponsePort* (vedi sezione 4.4.17.6);
- `reset(): GetEmailMethodResponseBuilderPort`
  Esegue il reset del builder.

4.4.17.5 QueryEmailMethodResponseBuilderAdapter

Implementa l'interfaccia *QueryEmailMethodResponseBuilderPort* (sezione 4.4.17.4).

Attributi:

- `queryEmailMethodResponseBuilder: QueryEmailMethodResponseBuilder`
  Adaptee dell'adapter, classe della libreria JMAP che implementa il pattern builder per costruire la risposta alla chiamata JMAP Email/Query.

4.4.17.6 QueryEmailMethodResponsePort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare la risposta JMAP della chiamata Email/Query.

4.4.17.7 QueryEmailMethodResponseAdapter

Implementa l'interfaccia *QueryEmailMethodResponsePort* (paragrafo 4.4.17.6).

Attributi:

- `queryEmailMethodResponse: QueryEmailMethodResponse`
  Adaptee dell'adapter, classe della libreria JMAP che modella una risposta alla chiamata JMAP Email/Query.

Metodi:

- `adaptee(): QueryEmailMethodResponse`
  Ritorna l'adaptee.

4.4.17.8 QueryEmailMethodCallPort

Interfaccia che definisce il contratto che deve avere l'adapter che vuole modellare una chiamata JMAP Email/Query.

4.4.17.9 QueryEmailMethodCallAdapter

Implementa l'interfaccia *QueryEmailMethodCallPort* (paragrafo 4.4.17.8).

Attributi:

- `queryEmailMethodCall: QueryEmailMethodCall`
  Adaptee dell'adapter, classe della libreria JMAP che modella una chiamata JMAP Email/Query.

Metodi:

- `adaptee(): QueryEmailMethodCall`
  Ritorna l'adaptee.

4.4.18 Classi entità JMAP

Classi e porte che rappresentano l'entità JMAP definite nello standard.

4.4.18.1 AccountState

Rappresenta lo stato di un account JMAP.

Attributi:

- `id: String`
  Id dell'account;
- `state: String`
  Stato attuale dell'account.

Metodi:

- `ConstructorByField`
  Costruttore;
- `AccountState(id: String)`
  Costruttore per un nuovo AccountState che parte dallo stato zero;
- `id(): String`
  Ritorna l'id dell'account;
- `state(): String`
  Ritorna lo stato dell'account;
- `increaseState(): AccountState`
  Crea una copia dell'oggetto con lo stato aumentato di uno.

4.4.18.2 EmailPort

Dichiara il contratto che deve avere la classe che rappresenta l'oggetto di dominio: email.

Metodi:

- `getId(): String`
  Ritorna l'id dell'email;
- `getBlobId(): String`
  Ritorna l'id del blob dell'email;

- `getKeywords(): Map<String, Boolean>`
  Ritorna una mappa di valori stringhe-booleani delle keywords dell'email;
- `getMailboxIds(): Map<String, Boolean>`
  Ritorna una mappa valori stringhe-booleani degli id delle mailbox dell'email;
- `getThreadId(): String`
  Ritorna l'id del thread dell'email;
- `getSize(): Long`
  Ritorna la dimensione dell'email;
- `getAttachments(): List<EmailAttachment>`
  Ritorna la lista degli allegati dell'email;
- `getBodyValues(): Map<String, EmailBodyValuePort>`
  Ritorna la mappa valori dei body parts dell'email;
- `getReceivedAt(): Instant`
  Ritorna la data di ricezione dell'email;
- `getSubject(): String`
  Ritorna l'oggetto dell'email;
- `getMessageId(): List<String>`
  Ritorna la lista degli id dei messaggi dell'email;
- `getInReplyToMessageId(): List<String>`
  Ritorna la lista degli id dei messaggi a cui l'email risponde;
- `getReferences(): List<String>`
  Ritorna la lista degli id dei messaggi a cui l'email fa riferimento;
- `getReplyTo(): List<EmailAddressPort>`
  Ritorna la lista degli indirizzi email a cui rispondere;
- `getSender(): List<EmailAddressPort>`
  Ritorna la lista degli indirizzi email del mittente effettivo;
- `getTo(): List<EmailAddressPort>`
  Ritorna la lista degli indirizzi email dei destinatari;

- `getFrom(): List<EmailAddressPort>`
  Ritorna la lista degli indirizzi email dei mittenti;
- `getTextBody(): List<EmailBodyPartPort>`
  Ritorna la lista dei body parts di tipo testo;
- `getHtmlBody(): List<EmailBodyPartPort>`
  Ritorna la lista dei body parts di tipo html;
- `getSentAt(): OffsetDateTime`
  Ritorna la data di invio dell'email;
- `getBodyStructure(): EmailBodyPartPort`
  Ritorna la struttura del body dell'email;
- `getHeaders(): List<EmailHeaderPort>`
  Ritorna la lista degli header dell'email;
- `toBuiler(): EmailBuilderPort`
  Ritorna un builder per costruire un'email.

4.4.18.3 EmailAdapter

Classe che implementa l'interfaccia *EmailPort*.

Attributi:

- `email: Email`
  Email da rappresentare.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.18.4 EmailBodyPartPort

Dichiara il contratto che deve avere la classe che si occupa del body part di un'email.

Metodi:

- `getPartId(): String`
  Ritorna l'id del body part;
- `getType(): String`
  Ritorna il tipo del body part;
- `getSize(): Long`
  Ritorna la dimensione del body part;
- `getName(): String`
  Ritorna il nome del body part;
- `toBuilder(): EmailBodyPartBuilderPort`
  Ritorna un builder già configurato con i campi dell'istanza BodyPart per costruire un body part modificato.

4.4.18.5 EmailBodyPartAdapter

Classe che implementa l'interfaccia *EmailBodyPartPort*.

Attributi:

- `emailBodyPart: EmailBodyPart`
  Body part da rappresentare.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.18.6 MailboxPort

Dichiara il contratto che deve avere la classe che si occupa della mailbox.

Metodi:

- `getId(): String`
  Ritorna l'id della mailbox;

- `getName(): String`
  Ritorna il nome della mailbox;
- `getRole(): String`
  Ritorna il ruolo della mailbox;
- `getParentId(): String`
  Ritorna l'id della mailbox genitore;
- `getSortOrder(): Long`
  Ritorna l'ordine della mailbox (la UI del client usa questa per ordinare le mailbox);
- `getTotalEmails(): Long`
  Ritorna il numero totale di email nella mailbox;
- `getUnreadEmails(): Long`
  Ritorna il numero di email non lette nella mailbox;
- `getTotalThreads(): Long`
  Ritorna il numero totale di thread nella mailbox;
- `getUnreadThreads(): Long`
  Ritorna il numero di thread non letti nella mailbox;
- `getMyRights(): MailboxRightsPort`
  Ritorna i diritti dell'utente sulla mailbox (ACL_G);
- `getIsSubscribed(): Boolean`
  Ritorna vero se l'utente è iscritto alla mailbox;
- `toBuilder(): MailboxBuilderPort`
  Ritorna un builder già configurato con i campi dell'istanza mailbox per costruire una mailbox modificata.

4.4.18.7 SessionResourcePort

Dichiara il contratto che deve avere la classe che rappresenta la risorsa di sessione.

Metodi:

- `username(): String`
  Ritorna l'username della sessione;
- `accounts(): Map<String, AccountPort>`
  Ritorna una mappa di account degli account collegati all'utente;
- `primaryAccount(): Map<ClassAccountCapabilityPort, String>`
  Ritorna l'account primario per ogni capability del server per l'utente;
- `state(): String`
  Ritorna lo stato della sessione.

4.4.18.8 EmailChangesTracker

Dichiara il contratto che deve avere la classe che si occupa di tracciare i cambiamenti delle email.

Metodi:

- `id(): String`
  Ritorna l'id dell'email;
- `created(): Map<String, String>`
  Ritorna una mappa di email create;
- `updated(): Map<String, String>`
  Ritorna una mappa di email aggiornate;
- `destroyed(): Map<String, String>`
  Ritorna una mappa di email distrutte;
- `emailHasBeenCreated(newState: String, emailId: String): EmailChangesTracker`
  Aggiunge una email creata;
- `emailHasBeenUpdated(newState: String, emailId: String): EmailChangesTracker`
  Aggiunge una email aggiornata;
- `emailHasBeenDestroyed(emailId: String): EmailChangesTracker`
  Aggiunge una email distrutta.

4.4.18.9 SimpleEmailChangesTracker

Implementazione di *EmailChangesTracker* (paragrafo 4.4.18.8) semplificata rispetto allo standard JMAP.

Attributi:

- `id: String`
  Id dell'account;
- `created: Map<String, String>`
  Mappa che segna ogni stato in cui è stata creata una email e l'id di quest'ultima;
- `updated: Map<String, String>`
  Mappa che segna ogni stato in cui è stata aggiornata una email e l'id di quest'ultima;
- `destroyed: Map<String, String>`
  Mappa che segna ogni stato in cui è stata distrutta una mailbox e l'id di quest'ultima.

4.4.18.10 MailboxChangesTracker

Dichiara il contratto che deve avere la classe che si occupa di tracciare i cambiamenti delle mailbox.

Metodi:

- `id(): String`
  Ritorna l'id della mailbox;
- `created(): Map<String, String>`
  Ritorna una mappa di mailbox create;
- `updated(): Map<String, String>`
  Ritorna una mappa di mailbox aggiornate;
- `destroyed(): Map<String, String>`
  Ritorna una mappa di mailbox distrutte;

- `mailboxHasBeenCreated(newState: String, mailboxId: String): MailboxChangesTracker`
  Aggiunge una mailbox creata;
- `mailboxHasBeenUpdated(newState: String, mailboxId: String): MailboxChangesTracker`
  Aggiunge una mailbox aggiornata;
- `mailboxHasBeenDestroyed(mailboxId: String): MailboxChangesTracker`
  Aggiunge una mailbox distrutta.

4.4.18.11 SimpleMailboxChangesTracker

Implementazione di *MailboxChangesTracker* (paragrafo 4.4.18.10) semplificata rispetto allo standard JMAP.

Attributi:

- `id: String`
  Id dell'account;
- `created: Map<String, String>`
  Mappa che segna ogni stato in cui è stata creata una mailbox e l'id di quest'ultima;
- `updated: Map<String, String>`
  Mappa che segna ogni stato in cui è stata aggiornata una mailbox e l'id di quest'ultima;
- `destroyed: Map<String, String>`
  Mappa che segna ogni stato in cui è stata distrutta una mailbox e l'id di quest'ultima.

4.4.18.12 IdentityChangesTracker

Dichiara il contratto che deve avere la classe che si occupa di tracciare i cambiamenti delle identità.

Metodi:

- `id(): String`
  Ritorna l'id dell'identità;
- `created(): Map<String, String>`
  Ritorna una mappa di identità create;
- `updated(): Map<String, String>`
  Ritorna una mappa di identità aggiornate;
- `destroyed(): Map<String, String>`
  Ritorna una mappa di identità distrutte;
- `identityHasBeenCreated(newState: String, identityId: String): IdentityChangesTracker`
  Aggiunge un'identità creata;
- `identityHasBeenUpdated(newState: String, identityId: String): IdentityChangesTracker`
  Aggiunge un'identità aggiornata;
- `identityHasBeenDestroyed(identityId: String): IdentityChangesTracker`
  Aggiunge un'identità distrutta.

4.4.18.13 SimpleIdentityChangesTracker

Implementazione di *IdentityChangesTracker* (paragrafo 4.4.18.12) semplificata rispetto allo standard JMAP.

Attributi:

- `id: String`
  Id dell'account;
- `created: Map<String, String>`
  Mappa che segna ogni stato in cui è stata creata una identity e l'id di quest'ultima;
- `updated: Map<String, String>`
  Mappa che segna ogni stato in cui è stata aggiornata una identity e l'id di quest'ultima;

- `destroyed: Map<String, String>`
  Mappa che segna ogni stato in cui è stata distrutta una identity e l'id di quest'ultima.

4.4.18.14 ThreadChangesTracker

Dichiara il contratto che deve avere la classe che si occupa di tracciare i cambiamenti dei thread.

Metodi:

- `id(): String`
  Ritorna l'id del thread;
- `created(): Map<String, String>`
  Ritorna una mappa di thread creati;
- `updated(): Map<String, String>`
  Ritorna una mappa di thread aggiornati;
- `destroyed(): Map<String, String>`
  Ritorna una mappa di thread distrutti;
- `threadHasBeenCreated(newState: String, threadId: String): ThreadChangesTracker`
  Aggiunge un thread creato;
- `threadHasBeenUpdated(newState: String, threadId: String): ThreadChangesTracker`
  Aggiunge un thread aggiornato;
- `threadHasBeenDestroyed(newState: String, threadId: String): ThreadChangesTracker`
  Aggiunge un thread distrutto.

4.4.18.15 SimpleThreadChangesTracker

Implementazione di *ThreadChangesTracker* (paragrafo 4.4.18.14) semplificata rispetto allo standard JMAP.

Attributi:

- `id: String`
  Id dell'account;
- `created: Map<String, String>`
  Mappa che segna ogni stato in cui è stato creato un thread e l'id di quest'ultimo;
- `updated: Map<String, String>`
  Mappa che segna ogni stato in cui è stato aggiornato un thread e l'Id di quest'ultimo;
- `destroyed: Map<String, String>`
  Mappa che segna ogni stato in cui è stato distrutto un thread e l'Id di quest'ultimo.

4.4.18.16 ReferenceIdsResolverPort

Dichiara il contratto che deve avere la classe che si occupa di risolvere gli id di riferimento.

Metodi:

- `resolve(resultReferenceResolverPort: ResultReferenceResolverPort, previousResponses: ListMultimapPort<String, ResponseInvocationPort>): String[0..*]`
  Risolve gli id di riferimento.

4.4.18.17 ReferenceIdsResolverAdapter

Classe che implementa l'interfaccia *ReferenceIdsResolverPort*.

Attributi:

- `referenceIdsResolverPort : ReferenceIdsResolver`
  Risolutore di id di riferimento da rappresentare della libreria JMAP.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.18.18 IdentityPort

Dichiara il contratto che deve avere la classe che rappresenta l'identità.

Metodi:

- `getId(): String`
  Ritorna l'id dell'identità;
- `getName(): String`
  Ritorna il nome dell'identità;
- `getEmail(): String`
  Ritorna l'email dell'identità;
- `getReplyTo(): EmailAddressPort[0..*]`
  Ritorna gli indirizzi email a cui rispondere;
- `getBcc(): EmailAddressPort[0..*]`
  Ritorna gli indirizzi email in copia nascosta;
- `getTextSignature(): String`
  Ritorna la firma in testo;
- `getHtmlSignature(): String`
  Ritorna la firma in html;
- `getMayDelete(): Boolean`
  Ritorna vero se l'utente può cancellare l'identità;
- `toBuilder(): IdentityBuilderPort`
  Ritorna un builder per costruire un'identità, il builder copia i campi già impostati dell'istanza su cui il metodo è stato chiamato.

4.4.18.19 ThreadPort

Dichiara il contratto che deve avere la classe che rappresenta il thread.

Metodi:

- `getId(): String`
  Ritorna l'id del thread;

- `getEmailIds(): List<String>`
  Ritorna la lista degli id delle email;
- `toBuilder(): ThreadBuilderPort`
  Ritorna un builder per costruire un thread.

4.4.18.20 SetErrorPort

Definisce il contratto che deve avere l'adapter per rappresentare un errore in una chiamata JMAP /Set generica.

4.4.18.21 SetErrorAdapter

Implementazione di *SetErrorPort* (paragrafo 4.4.18.20) .

Attributi:

- `setError: SetError`
  Adaptee dell'adapter, classe della libreria JMAP che rappresenta un errore in una chiamata JMAP /Set generica.

Metodi:

- `adaptee(): SetError`
  Ritorna l'adaptee.

4.4.18.22 setErrorEnumPort

Definisce il contratto che deve avere l'adapter che implementa la possibilità di creare dei *SetErrorPort* (paragrafo 4.4.18.20) .

Metodi:

- `singleton(): SetErrorPort`
  Ritorna l'errore SetErrorPort singleton;
- `invalidProperties(): SetErrorPort`
  Ritorna l'errore SetErrorPort invalidProperties;
- `notFound(): SetErrorPort`
  Ritorna l'errore SetErrorPort notFound;

- `invalidPatch(): SetErrorPort`
  Ritorna l'errore SetErrorPort invalidPatch.

4.4.18.23 IfInStateMatch

Definisce l'interfaccia che deve avere la classe che assolve al compito di controllare se lo stato passato corrisponde a quello attuale del server.

Metodi:

- `methodStateMatchCurrent(methodCallState: String, currentObjectState: String): void`
  Controlla se c'è un mismatch e se c'è lancia l'eccezione *StateMismatchException* (paragrafo 4.4.20.7) .

4.4.18.24 StandardIfInStateMatch

Implementazione di *IfInStateMatch* (paragrafo 4.4.18.23) come da standard JMAP.

4.4.18.25 CreatedResult<EntityType>

Record che rappresenta un generico risultato durante la creazione di un tipo di entità.

Attributi/Metodi (record):

- `created: Map<String, EntityType>`
  Risultato dell'entità create;
- `notCreated: Map<String, SetErrorPort>`
  Risultato dell'entità non create con il relativo errore.

4.4.18.26 UpdatedResult<EntityType>

Record che rappresenta un generico risultato durante l'update di un tipo di entità.

Attributi/Metodi (record):

- `updated: Map<String, EntityType>`
  Risultato dell'entità aggiornante;
- `notUpdated: Map<String, SetErrorPort>`
  Risultato dell'entità non aggiornante con il relativo errore.

4.4.18.27 DestroyedResult

Record che rappresenta un generico risultato durante la creazione di un tipo di entità.

Attributi/Metodi (record):

- `destroyed: String[0..*]`
  Risultato dell'entità distrutte;
- `notDestroyed: Map<String, SetErrorPort>`
  Risultato dell'entità non distrutte con il relativo errore.

4.4.18.28 GetMethodCallPort

Dichiara il contratto che deve avere la classe che si occupa di rappresentare la chiamata generica JMAP /Get.

Metodi:

- `accountId(): String`
  Ritorna l'id dell'account;
- `getIds(): String[0..*]`
  Ritorna gli id;
- `getProperties(): String[0..*]`
  Ritorna le proprietà;
- `getIdsReference(): InvocationResultReferencePort`
  Ritorna gli id di riferimento.

4.4.18.29 ChangesMethodCallPort

Dichiara il contratto che deve avere la classe che si occupa di rappresentare la chiamata generica JMAP /Changes.

Metodi:

- `accountId(): String`
  Ritorna l'accountId della richiesta;

- `getSinceState(): String`
  Ritorna lo stato da cui bisogna iniziare a calcolare tutti i cambiamenti;
- `getMaxChanges(): Long`
  Ritorna il massimo numero di changes che la risposta deve tornare.

4.4.19 Classi per la persistenza

Classi e porte per la persistenza dei dati.

4.4.19.1 AccountStateRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza dello stato dell'account.

Metodi:

- `retrieve(accountId: String): AccountState`
  Ritorna lo stato dell'account;
- `save(accountState: AccountState): void`
  Salva lo stato dell'account.

4.4.19.2 AccountStateRepositoryAdapter

Implementa l'interfaccia *AccountStateRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

4.4.19.3 EmailRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza delle email.

Metodi:

- `retrieveAll(accountId: String): retrievedEntity<EmailPort>`
  Ritorna tutte le email dell'account;
- `retrieve(ids: String[0..*]): retrievedEntity<EmailPort>`
  Ritorna le email con gli ids specificati;
- `retrieveOne(emailId: String): EmailPort`
  Ritorna l'email con l'id specificato;
- `destroy(emailId: String): void`
  Elimina l'email con l'id specificato;
- `save(emailPort: EmailPort): void`
  Salva l'email specificata;
- `overwrite(emailPort: EmailPort): void`
  Aggiorna l'email specificata.

4.4.19.4 EmailRepositoryAdapter

Implementa l'interfaccia *EmailRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.5 MailboxRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza delle mailbox.

Metodi:

- `retrieveAll(accountId: String): retrievedEntity<MailboxPort>`
  Ritorna tutte le mailbox dell'account;
- `retrieve(ids: String[0..*]): retrievedEntity<MailboxPort>`
  Ritorna le mailbox con gli ids specificati;
- `retrieveOne(mailboxId: String): MailboxPort`
  Ritorna la mailbox con l'id specificato;
- `destroy(mailboxId: String): void`
  Elimina la mailbox con l'id specificato;
- `save(mailboxPort: MailboxPort): void`
  Salva la mailbox specificata;
- `overwrite(mailboxPort: MailboxPort): void`
  Aggiorna la mailbox specificata.

4.4.19.6 MailboxRepositoryAdapter

Implementa l'interfaccia *MailboxRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.7 IdentityRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza delle identità.

Metodi:

- `retrieveAll(accountId: String): retrievedEntity<IdentityPort>`
  Ritorna tutte le identità dell'account;
- `retrieve(ids: String[0..*]): retrievedEntity<IdentityPort>`
  Ritorna le identità con gli ids specificati;
- `destroy(identityId: String): void`
  Elimina l'identità con l'id specificato;
- `save(identityPort: IdentityPort): void`
  Salva l'identità specificata.

4.4.19.8 IdentityRepositoryAdapter

Implementa l'interfaccia *IdentityRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.9 ThreadRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza dei thread.

Metodi:

- `retrieveAll(accountId: String): retrievedEntity<ThreadPort>`
  Ritorna tutti i thread dell'account;
- `retrieve(ids: String[0..*]): retrievedEntity<ThreadPort>`
  Ritorna i thread con gli ids specificati;
- `retrieveOne(threadId: String): ThreadPort`
  Ritorna il thread con l'id specificato;
- `save(threadPort: ThreadPort): void`
  Salva il thread specificato.

4.4.19.10 ThreadRepositoryAdapter

Implementa l'interfaccia *ThreadRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.11 EmailChangesTrackerRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza del tracker delle modifiche delle email dell'account.

Metodi:

- `retrieve(accountId: String): EmailChangesTracker`
  Ritorna lo stato del tracker delle modifiche delle email dell'account;
- `save(emailChangesTracker: EmailChangesTracker): void`
  Salva lo stato del tracker delle modifiche delle email dell'account.

4.4.19.12 EmailChangesTrackerRepositoryAdapter

Implementa l'interfaccia *EmailChangesTrackerRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.13 MailboxChangesTrackerRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza del tracker delle modifiche delle mailbox dell'account.

Metodi:

- `retrieve(accountId: String): MailboxChangesTracker`
  Ritorna lo stato del tracker delle modifiche delle mailbox dell'account;
- `save(mailboxChangesTracker: MailboxChangesTracker): void`
  Salva lo stato del tracker delle modifiche delle mailbox dell'account.

4.4.19.14 MailboxChangesTrackerRepositoryAdapter

Implementa l'interfaccia *MailboxChangesTrackerRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.15 IdentityChangesTrackerRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza del tracker delle modifiche delle identità dell'account.

Metodi:

- `retrieve(accountId: String): IdentityChangesTracker`
  Ritorna lo stato del tracker delle modifiche delle identità dell'account;
- `save(identityChangesTracker: IdentityChangesTracker): void`
  Salva lo stato del tracker delle modifiche delle identità dell'account.

4.4.19.16 IdentityChangesTrackerRepositoryAdapter

Implementa l'interfaccia *IdentityChangesTrackerRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;
- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.17 ThreadChangesTrackerRepository

Dichiara il contratto che deve avere la classe che si occupa della persistenza del tracker delle modifiche dei thread dell'account.

Metodi:

- `retrieve(accountId: String): ThreadChangesTracker`
  Ritorna lo stato del tracker delle modifiche dei thread dell'account;
- `save(threadChangesTracker: ThreadChangesTracker): void`
  Salva lo stato del tracker delle modifiche dei thread dell'account.

4.4.19.18 ThreadChangesTrackerRepositoryAdapter

Implementa l'interfaccia *ThreadChangesTrackerRepository*.

Attributi:

- `COLLECTION: String`
  Nome della collezione;

- `mongoConnection: MongoConnection`
  Connessione al database;
- `gson: Gson`
  Libreria per la serializzazione e deserializzazione di oggetti Java.

Metodi:

- `ConstructorByField`
  Costruttore.

4.4.19.19 MongoConnection

Attributi:

- `jmapDatabase: MongoDatabase`
  Database JMAP.

Metodi:

- `ConstructorByField`
  Costruttore;
- `getDatabase(): MongoDatabase`
  Ritorna il database JMAP.

4.4.20 Eccezioni

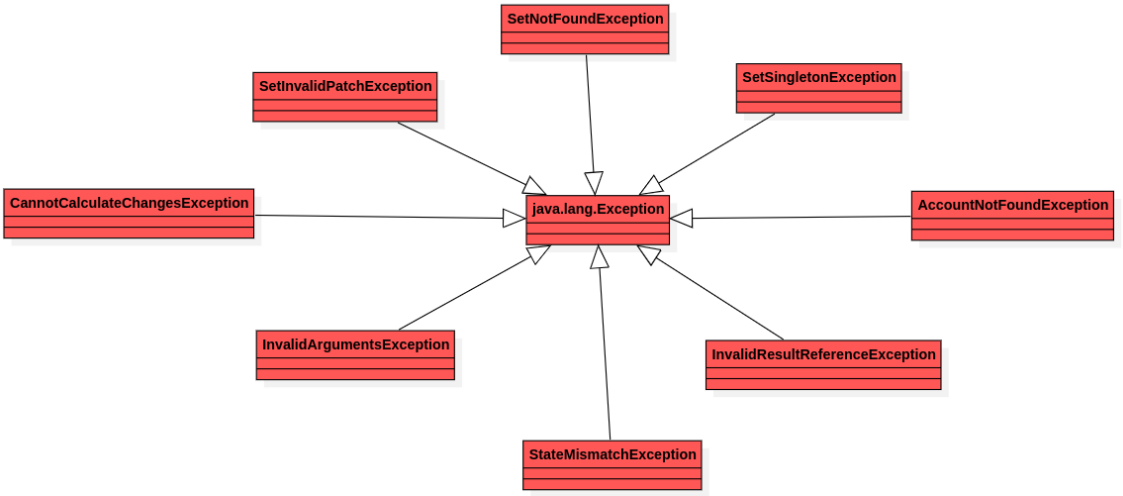

Figura 20: Diagramma delle eccezioni.

Tutte le eccezioni definite estendo semplicemente *java.lang.Exception*.

4.4.20.1 CannotCalculateChangesException

Errore JMAP: il server non è capace di calcolare i cambiamenti all'entità come richiesto.

4.4.20.2 SetInvalidPatchException

Errore JMAP: l'*ObjectPatch* passato non è una patch valida.

4.4.20.3 SetNotFoundException

Errore JMAP: l'entità non può essere recuperata perché non esiste.

4.4.20.4 SetSingletonException

Errore JMAP: l'entità non può essere creata e salvata perché già esiste.

4.4.20.5 AccountNotFoundException

Errore: non è stato trovato l' *AccountState* (paragrafo 4.4.18.1) indicato.

4.4.20.6 InvalidResultReferenceException

Errore JMAP: l'id reference indicato da client non è mai stato risolto dal server.

4.4.20.7 StateMismatchException

Errore JMAP: lo stato indicato dalla chiamata non corrisponde a quello attuale del server.

4.4.20.8 InvalidArgumentsException

Errore JMAP: l'argomento passato non è valido.

5 Stato requisiti

Di seguito trascriviamo ciascun requisito riportato nel documento *Analisi dei Requisiti v2.0.0* sezione Requisiti Funzionali, con accanto lo stato di implementazione del requisito nel prodotto.

Tabella 12: Requisiti funzionali

Codice	Importanza	Descrizione	Stato
RFO - 1	Obbligatorio	È necessario che il MUA abbia la capacità di inviare un'e-mail	Implementato
RFO - 1.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id dell'account durante l'attività di invio e-mail	Implementato
RFO - 1.2	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id dell'e-mail durante l'attività di invio e-mail	Implementato
RFO - 1.3	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il destinatario dell'e-mail durante l'attività di invio e-mail	Implementato
RFO - 1.4	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il mittente dell'e-mail durante l'attività di invio e-mail	Implementato
RFO - 1.5	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se l'id e-mail non è valido durante l'attività di invio e-mail	Implementato

RFO - 1.6	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se l'oggetto non è valido durante l'attività di invio e-mail	Implementato
RFO - 1.7	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se il corpo non è valido durante l'attività di invio e-mail	Implementato
RFO - 1.8	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se il destinatario non è valido durante l'attività di invio e-mail	Implementato
RFO - 1.9	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se il destinatario non esiste durante l'attività di invio e-mail	Implementato
RFO - 1.10	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se sono presenti troppi destinatari durante l'attività di invio e-mail	Implementato
RFO - 1.11	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se il mittente non è valido durante l'attività di invio e-mail	Implementato
RFO - 2	Obbligatorio	È necessario che il MUA abbia la capacità di creare un'e-mail	Implementato
RFO - 2.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il destinatario dell'e-mail durante l'attività di creazione e-mail	Implementato

RFO - 2.2	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il mittente dell'e-mail durante l'attività di creazione e-mail	Implementato
RFO - 2.3	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'oggetto dell'e-mail durante l'attività di creazione e-mail	Implementato
RFO - 2.4	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il corpo dell'e-mail durante l'attività di creazione e-mail	Implementato
RFO - 2.5	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella dell'e-mail durante l'attività di creazione e-mail	Implementato
RFO - 2.6	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se le cartelle di destinazione sono troppe	Implementato
RFO - 3	Obbligatorio	È necessario che il MUA abbia la capacità di creare una cartella	Implementato
RFO - 3.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il nome della cartella durante l'attività di creazione cartella	Implementato
RFO - 3.2	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella genitore durante l'attività di creazione cartella	Implementato

RFO - 3.3	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se si crea una cartella con un nome non valido durante l'attività di creazione cartella	Implementato
RFO - 3.4	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se non si trova la cartella genitore durante l'attività di creazione cartella	Implementato
RFO - 3.5	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se si crea una cartella duplicata durante l'attività di creazione cartella	Implementato
RFD - 4	Desiderabile	È necessario che il MUA abbia la capacità di creare una condivisione di una cartella	Non implementato
RFD - 4.1	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella durante l'attività di creazione condivisione cartella	Non implementato
RFD - 4.2	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'indirizzo contatto durante l'attività di creazione condivisione cartella	Non implementato
RFD - 4.3	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'id cartella trasmesso durante l'attività di creazione condivisione cartella non è stato trovato	Non implementato

RFD - 4.4	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'indirizzo del contatto trasmesso durante l'attività di creazione condivisione cartella non è stato trovato	Non implementato
RFO - 5	Obbligatorio	È necessario che il MUA abbia la capacità di modificare un'e-mail	Implementato
RFO - 5.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il destinatario durante l'attività di modifica e-mail	Implementato
RFO - 5.2	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il mittente dell'e-mail durante l'attività di modifica e-mail	Implementato
RFO - 5.3	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'oggetto dell'e-mail durante l'attività di modifica e-mail	Implementato
RFO - 5.4	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il corpo dell'e-mail durante l'attività di modifica e-mail	Implementato
RFO - 5.5	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere la cartella dell'e-mail durante l'attività di modifica e-mail	Implementato
RFO - 5.6	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se le cartelle di destinazione sono troppe	Implementato

RFO - 6	Obbligatorio	È necessario che il MUA abbia la capacità di modificare una cartella	Implementato
RFO - 6.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere il nome della cartella durante l'attività di modifica cartella	Implementato
RFO - 6.2	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella genitore durante l'attività di modifica cartella	Implementato
RFO - 6.3	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se viene trasmesso un nome non valido durante l'attività di modifica cartella	Implementato
RFO - 6.4	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se non si trova la cartella genitore durante l'attività di modifica cartella	Implementato
RFO - 6.5	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore nel caso in cui venga creata una cartella duplicata durante l'attività di modifica cartella	Implementato
RFD - 7	Desiderabile	È necessario che il MUA abbia la capacità di modificare una condivisione di una cartella	Non implementato

RFD - 7.1	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella durante l'attività di modifica condivisione cartella	Non implementato
RFD - 7.2	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'indirizzo contatto durante l'attività di modifica condivisione cartella	Non implementato
RFD - 7.3	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'id cartella trasmesso durante l'attività di modifica condivisione cartella non è stato trovato	Non implementato
RFD - 7.4	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'indirizzo e-mail del contatto trasmesso durante l'attività di modifica condivisione cartella non è valido	Non implementato
RFO - 8	Obbligatorio	È necessario che il MUA abbia la capacità di eliminare un'e-mail	Implementato
RFO - 8.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id dell'e-mail durante l'attività di eliminazione e-mail	Implementato
RFO - 8.2	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se l'id e-mail trasmesso durante l'attività di eliminazione e-mail non è stato trovato	Implementato

RFO - 9	Obbligatorio	È necessario che il MUA abbia la capacità di eliminare una cartella	Implementato
RFO - 9.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella durante l'attività di eliminazione cartella	Implementato
RFO - 9.2	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se l'id cartella trasmesso durante l'attività di eliminazione cartella non è stato trovato	Implementato
RFO - 9.3	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se all'interno della cartella sono presenti e-mail durante l'attività di eliminazione cartella	Implementato
RFO - 9.4	Obbligatorio	È necessario che il MUA visualizzi un messaggio d'errore se all'interno della cartella sono presenti altre cartelle durante l'attività di eliminazione cartella	Implementato
RFD - 10	Desiderabile	È necessario che il MUA abbia la capacità di eliminare una condivisione di una cartella	Non implementato
RFD - 10.1	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'id della cartella durante l'attività di eliminazione condivisione cartella	Non implementato

RFD - 10.2	Desiderabile	È necessario che il MUA abbia la capacità di trasmettere l'indirizzo contatto durante l'attività di eliminazione condivisione cartella	Non implementato
RFD - 10.3	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'id cartella trasmesso durante l'attività di eliminazione condivisione cartella non è stato trovato	Non implementato
RFD - 10.4	Desiderabile	È necessario che il MUA visualizzi un messaggio d'errore se l'indirizzo e-mail del contatto trasmesso durante l'attività di eliminazione condivisione cartella non è valido	Non implementato
RFZ - 11	Opzionale	È necessario che il MUA abbia la capacità di sincronizzare le e-mail	Implementato
RFZ - 11.1	Opzionale	È necessario che il MUA abbia la capacità di trasmettere l'id account durante l'attività di sincronizzazione e-mail	Implementato
RFZ - 11.2	Opzionale	È necessario che il MUA abbia la capacità di trasmettere l'id delle e-mail da aggiornare durante l'attività di sincronizzazione e-mail	Implementato
RFZ - 11.3	Opzionale	È necessario che il MUA abbia la capacità di trasmettere le proprietà delle e-mail da aggiornare durante l'attività di sincronizzazione e-mail	Implementato

RFZ - 11.4	Opzionale	È necessario che il MUA visualizzi un messaggio d'errore se la richiesta è troppo grande durante l'attività di sincronizzazione e-mail	Implementato
RFZ - 11.5	Opzionale	È necessario che il MUA visualizzi un messaggio d'errore se la proprietà non è valida durante l'attività di sincronizzazione e-mail	Implementato
RFZ - 12	Opzionale	È necessario che il MUA abbia la capacità di sincronizzare le cartelle	Implementato
RFZ - 12.1	Opzionale	È necessario che il MUA abbia la capacità di trasmettere l'id account durante l'attività di sincronizzazione cartelle	Implementato
RFZ - 12.2	Opzionale	È necessario che il MUA abbia la capacità di trasmettere l'id delle cartelle da aggiornare durante l'attività di sincronizzazione cartelle	Implementato
RFZ - 12.3	Opzionale	È necessario che il MUA abbia la capacità di trasmettere le proprietà delle cartelle da aggiornare durante l'attività di sincronizzazione cartelle	Implementato
RFZ - 12.4	Opzionale	È necessario che il MUA visualizzi un messaggio d'errore se la richiesta è troppo grande durante l'attività di sincronizzazione cartelle	Implementato

RFZ - 12.5	Opzionale	È necessario che il MUA visualizzi un messaggio d'errore se la proprietà non è valida durante l'attività di sincronizzazione cartelle	Implementato

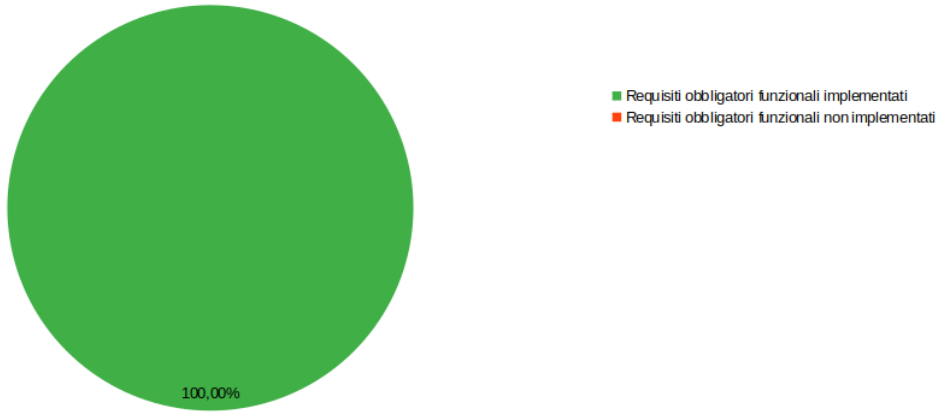

Figura 21: Grafico a torta del totale di requisti obbligatori funzionali implementati sul totale dei requisiti funzionali obbligatori.

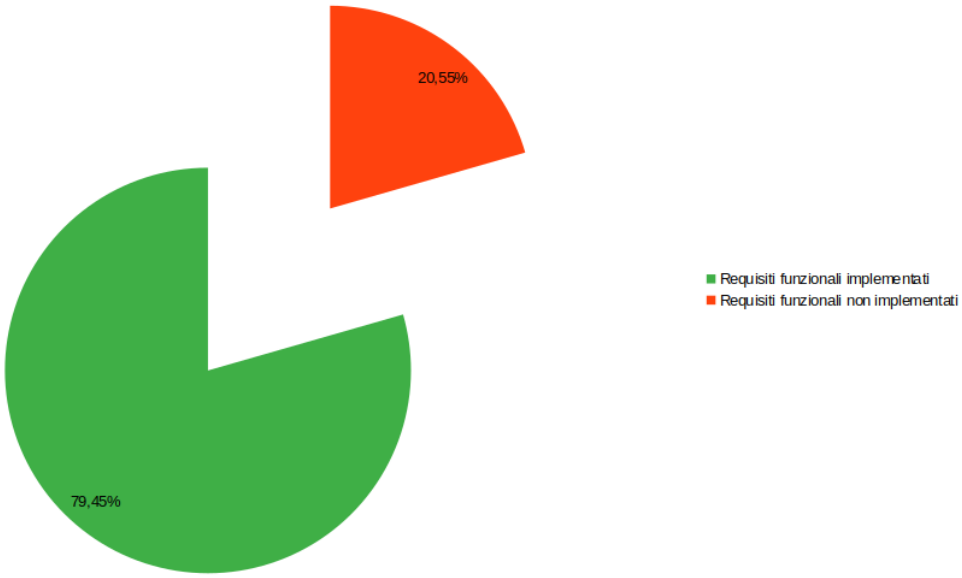

Figura 22: Grafico a torta del totale di requisti funzionali implementati sul totale dei requisiti funzionali.