Progetto:

BUDGETpal

Titolo del documento:

Documento di Architettura

Document Info:

Doc. Name	D3-BUDGETpal_DocumentoArchitettura	**Doc. Number**	**D3**
Description	Documento di Architettura descrivente la struttura, i componenti, le interazioni e le linee guida progettuali del sistema software.		

Indice

1. Scopo del documento

Il presente documento di architettura riporta una descrizione completa e dettagliata dell'architettura dell'applicazione web BUDGETpal.

In particolare, il documento include nella sua prima parte la definizione delle classi del sistema e la loro relativa descrizione con attributi e metodi principali.

Nel secondo capitolo viene presentata la descrizione dettagliata dei vincoli e delle regole applicate alle classi e alle loro interazioni. Il linguaggio OCL è utilizzato per specificare condizioni che devono essere rispettate dagli oggetti del sistema, in particolare dagli attributi e dai metodi, garantendo così la coerenza e la correttezza del modello.

Nella parte conclusiva del documento viene presentata una vista integrata dell'intera architettura. In un diagramma complessivo vengono riassunti tutti i componenti descritti precedentemente con i relativi vincoli in OCL. Questo diagramma fornisce una panoramica globale del sistema, evidenziando come i vari elementi si integrano tra loro per realizzare le funzionalità richieste.

*Nota: all'interno del diagramma delle classi sono riportate **tutte** le classi, ma durante le seguenti revisioni del design in fase dello sviluppo, sono state identificate alcune parti dello schema che possono essere semplificate per pura questione di vincoli di tempo e delle risorse.*

2. Definizione delle classi

Nel presente capitolo vengono rappresentiamo le classi del sistema applicativo. Ogni classe rappresenta un ambito operativo del nostro sistema con relative caratteristiche e i comportamenti.

Componenti principali di una classe consistono in:

- **Attributi** : ovvero le proprietà o dati con cui andare ad operare. Gli attributi definiscono lo stato degli oggetti della classe.
- **Metodi** : che rappresentano le operazioni o funzioni che gli oggetti della classe possono eseguire e verranno implementate a livello di codice. I metodi definiscono il comportamento della classe e possono manipolare gli attributi oppure interagire con altri oggetti.

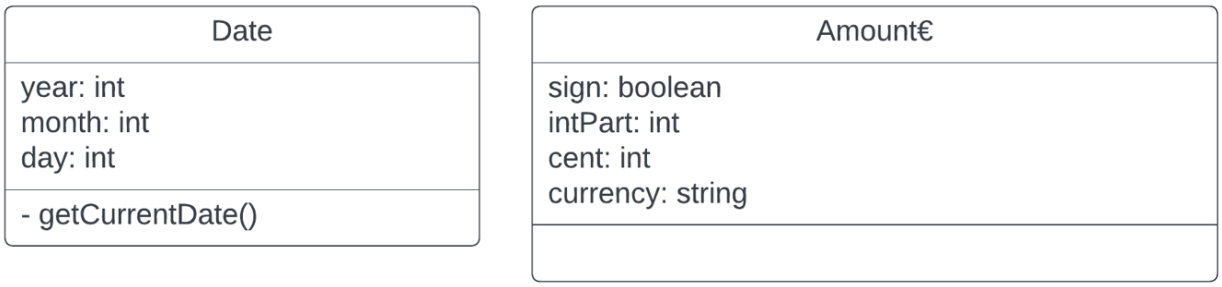
Di seguito viene offerta una breve descrizione di ogni classe, con la definizione degli attributi specifici e metodi di cui fanno uso.

Classi di supporto

Le classi di supporto sono chiamate così perché hanno lo scopo di definire dei tipi di dato necessari nelle altre classi.

La classe **Date** ci permette di rappresentare le date con il formato giorno/mese/anno.

La classe **Amount€** ci permette di rappresentare gli importi nel formato più comodo per la successiva elaborazione che divide l'importo in parte intera e centesimi con la possibilità anche di specificare la tipologia di transazione attraverso il segno e inserire la valuta dell'importo.

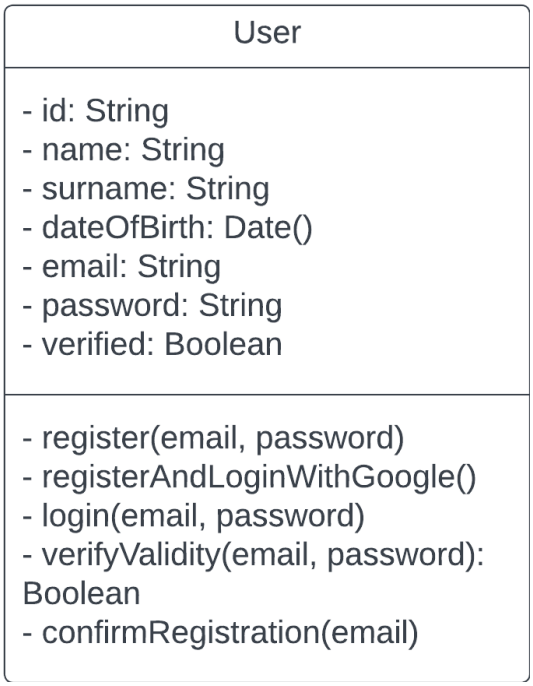


Classe User

La classe Utente viene utilizzata per salvare i dati relativi dell'utente registrato nel DB.

Gli attributi della classe utente sono:

- **Id:** il codice unico identificativo che viene dato ad ogni account per la richiesta di dati al DB
- **Name/Surname/DateofBirth/Email/Password**: sono gli attributi che vengono scelti dall'utente durante la registrazione.
- **Verified:** è un attributo che viene dato ad un user dopo aver verificato l'account attraverso la e-mail di verifica che viene inviata all'indirizzo e-mail inserito dall'utente

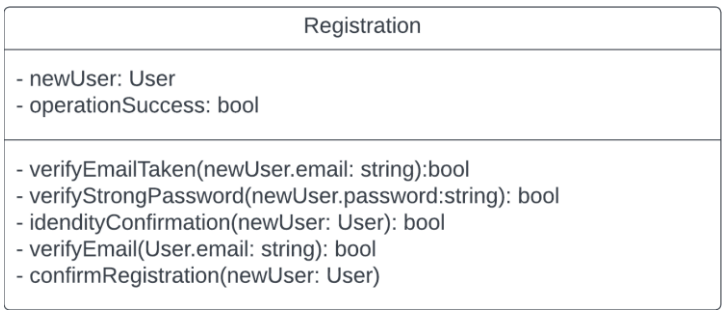


I metodi della classe utente sono:

- **Register:** questo metodo manda alla classe Registration i parametri e-mail e password.
- **RegisterandloginwithGoogle**: login/registrazione viene delegata al componente RegisterAndLoginWithGoogle che si occupa della formazione della richiesta di autenticazione Google.
- **Login**: questo metodo invia alla classe Login i parametri e-mail e password.
- **VerifyValidity:** l'informazione inserita in fase di login viene propagata al componente Login che effettua le verifiche sui credenziali inseriti.
- **ConfirmRegistration:** questo metodo ha l'obbiettivo di richiedere la conferma della registrazione al DB passando come parametro l'e-mail dell'utente

Classe Registration

La classe Registration viene utilizzata per fare le verifiche dovute in fase della prima registrazione dell'utente e salvare i dati relativi al nuovo utente nel DB.



Gli attributi della classe Registration sono:

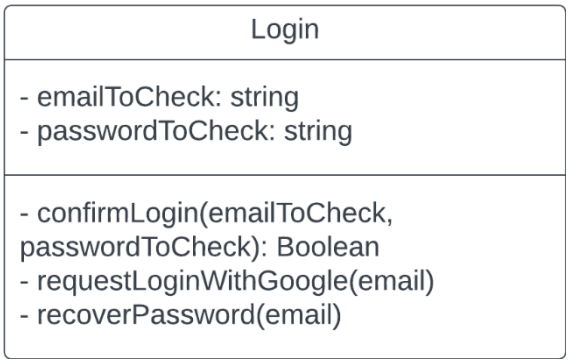
- **newUser**: in questo attributo vengono inseriti gli attributi della classe User nel DB per la creazione di un nuovo utente

- **operationSuccess**: l'esito registrazione viene dato dal DB dopo l'inserimento degli attributi dell'User e dopo aver verificato l'account attraverso la e-mail di verifica

I metodi della classe Registration sono:

- **verifyEmailTaken**: viene eseguita la ricerca nel DB dell'indirizzo email fornito dall'utente in fase di registrazione. Se tale email è stata già utilizzata da un altro utente, la registrazione viene abortita.
- **verifyStrongPassword**: viene eseguito il controllo della sicurezza della password forinita dall'utetne in fase di registrazione in base ai criteri prestabiliti.
- **identityConfirmation**
- **verifyEmail**: viene generato il messaggio per la conferma della registrazione il quale viene all'indirizzo destinazione.
- **confirmRegistration**: se vengono confermata l'esecuzione di tutti i controlli necessari per la registrazione viene abilitata la funzione di login.

Classe Login



Gli attributi della classe Login sono:

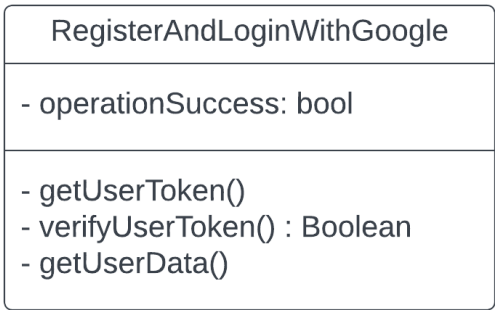
- **emailToCheck**
- **passwordToCheck**

Questi attributi permettono all'utente di inserire l'username e la password dell'account per eseguire il login

I metodi della classe Login sono:

- **confirmLogin**: questo metodo ha l'obbiettivo di verificare la correttezza dell'username e la password attraverso la Lettura del DB e in base al valore booleano restitutito permettere l'accesso all'applicazione in caso di esito positivo oppure negare l'accesso in caso di esito negativo.
- **RequestLoginWithGoogle**: in caso della richiesta di login/registrazione con account Google, il controllo viene delegato al componente RegisterAndLoginWithGoogle che si occupa della formazione della richiesta di autenticazione Google.
- **recoverPassword**: questo metodo permette l'invio di una password temporanea via e-mail per eseguire il login in modo temporaneo.

Classe registerAndLoginWithGoogle



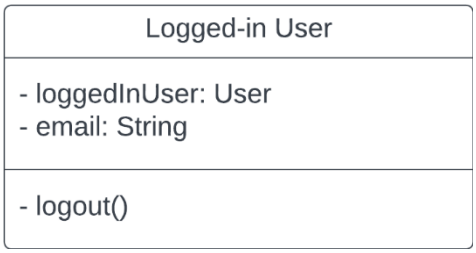
- Attributo **operationSuccess**: in questo attributo viene segnato l'esito del login/registrazione dalla verifica del corretto inserimento dei dati nella classe User attraverso una scrittura del DB

I metodi della classe comprendono seguenti funzioni:

- **getUserToken**: inizializza la richiesta di autenticazione ricevendo il token di accesso
- **verifyUserToken**: verifica la presenza del token nel DB
- **getUserData**: utilizzando il token di accesso, recupera i dati associati all'account google necessari per la creazione dell'istanza della classe User per il successivo salvataggio all'interno del DB

Classe Logged-in User

La classe Logged-in User è creata per dare accesso alle principali funzionalità dell'applicazione agli utenti che hanno completato il processo di login con successo. Gli attributi della classe logged-in user sono duplicati delle informazioni di User che ha effettuato accesso con email indicata nella fase di login:



- **loggedInUser**
- **email**

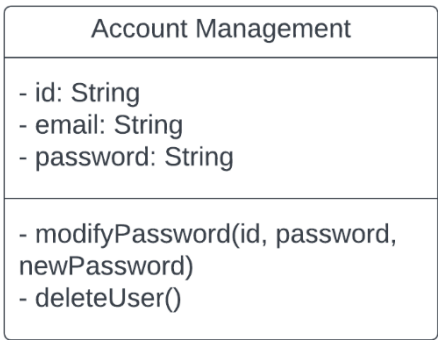
Il metodo della classe Logged-in User è:

- **logout**: questo metodo effettua logout e riporta l'utente alla pagina iniziale in cui è possibile eseguire il login o una nuova registrazione

Classe Account Management

Gli attributi della classe Account Management sono relative all'account con cui è stato effettuato login e corrispondono quindi agli attributi di Logged-in User:

- **Id**
- **Email**
- **Password**

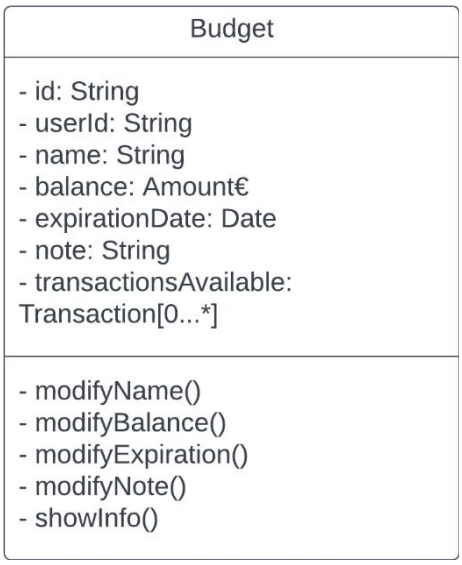


I metodi della classe Account Management sono:

- **modifyPassword**: questo metodo richiede all'utente gli attributi email e password per la scrittura nel DB di una nuova password da sovrascrivere alla password della classe User
- **deleteUser:** questo metodo permette di inizializzare il processo dell'eliminazione dell'account con relativa istanza di User ed esegue forzatamente logout.

Classe Budget

Gli attributi della classe Budget sono:



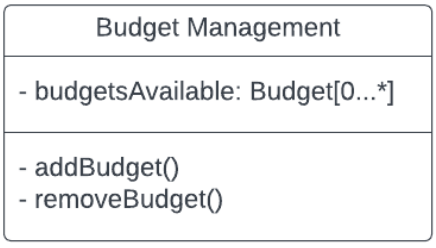
- **id**: il codice unico identificativo che viene dato ad ogni Budget
- **UserId**: attributo ID della classe User per identificare l'owner del Budget
- **Name/Balance/expirationDate/Note**: questi attributi vengono scelti dall'utente nel momento della creazione di un nuovo budget.
- **Transactions**: in questo attributo vengono segnati i dati segnati nella classe Transaction che successivamente vengono aggiunte o sottratti dal Balance.

I metodi della classe Budget sono:

- **ModifyName/ModififyBalance/modifyNote**: questi metodi permettono all'user di sovrascrivere gli attributi non unici del Budget all'interno del DB
- **deleteBudget**: questo metodo ha l'obbiettivo di eliminare completamente tutti i dati relativi al Budget all'interno del DB
- **showInfo**: questo metodo permette di mostrare a schermo tutte le informazioni relative al Budget scelto

Classe Budget Management

Gli attributi della classe Budget Management sono:



- **budgetsAvalible**: questo attributo rappresenta una lista di tutte le istanze della classe Budget collegate dallo stesso attributo UserID

I metodi della classe Budget Management sono:

- **addBudget**: questo metodo permette di creare un nuovo budget attraverso l'inserimento degli attributi della classe Budget
- **removeBudget**: questo metodo rimuove una classe Budget esistente all'interno del DB

Classe Recurrent Transaction / Non-recurrent Transaction

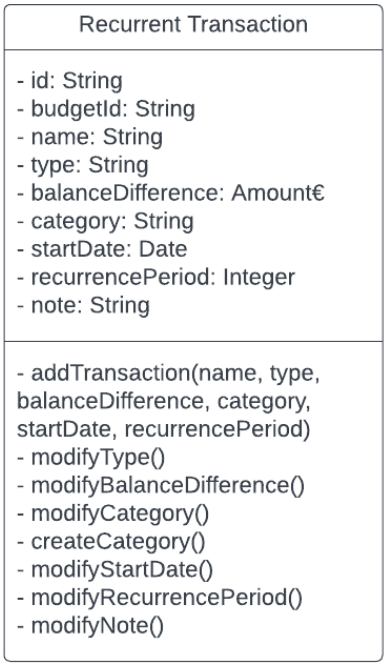
Gli attributi della classe [1]Recurrent Transaction / [2]Non-recurrent Transaction sono :

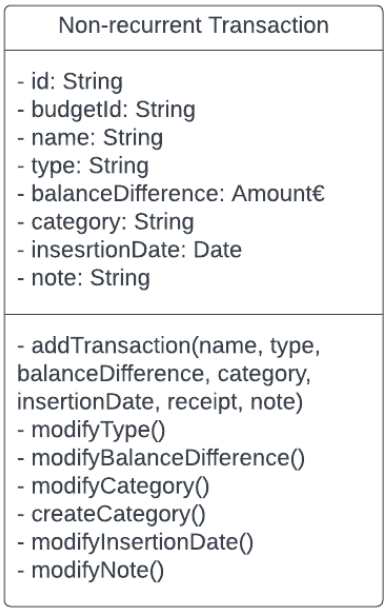
- **[1,2]Id**: numero unico e identificativo per ogni transazione
- **[1,2]BudgetID**: attributo della classe Budget per specificare a che budget appartiene
- **[1,2]name**: nome della transazione dato nella fase di creazione
- **[1,2]balanceDifference**: l'attributo che permette attraverso la classe di supporto Amount€ di inserire la quantità di denaro relativa alla transazione.
- **[1,2]category**: categoria in cui può essere classificata la transazione
- **[1,2]Note**: attributo con l'obbiettivo di descrivere la transazione.
- **[1]start Date**: inizio del periodo di una transazione ricorrente attraverso la classe di supporto Date.
- **[1]recurrencePeriod**: attributo che permette di inserire ogni quanto la transazione si ripete attraverso la classe di supporto Date.
- **[2]InsertionDate**: attributo che permette di inserire la data di quando avviene una transazione attraverso la classe di supporto Date.

i metodi della classe [1]Recurrent Transaction / [2]Non-recurrent Transaction sono:

- **[1,2]AddTransaction**
- **[1,2]ModifyType**
- **[1,2]modifyBalanceDifference**
- **[1,2]modifyCategory**
- **[1,2]createCategory**
- **[1]modifyStartDate**
- **[1]modifyRecurrentPeriod**
- **[2]modifyInsertionDate**
- **[1,2]modifyNote**

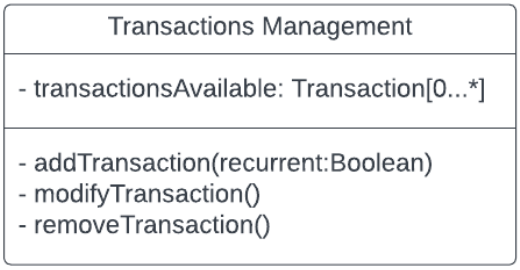
Tutti questi metodi hanno l'obbiettivo di creare o modificare dati specifici delle transazioni attraverso una scrittura nel DB





Classe Transactions Managment

Gli attributi della Classe Transactions Menagment sono :



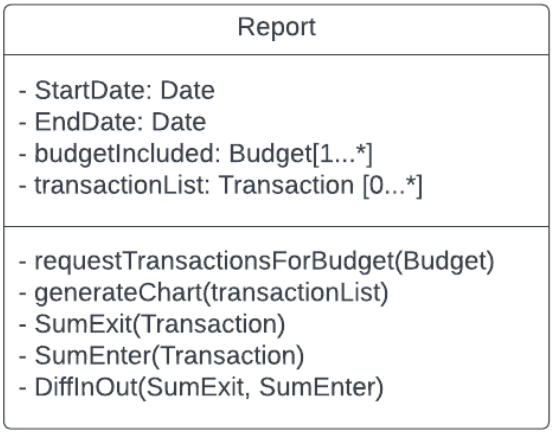
- **transactionsAvailable**: è una lista delle istanze della classe Transaction collegate dallo stesso attributo budgetId

i metodi della classe Transaction Menagment sono:

- **addTransaction**: questo metodo serve per la creazione di una nuova transazione
- **modifyTransaction/RemoveTransaction**: questi metodi permettno l'eliminazione o la modifica di una transazione

Classe Report

Gli attributi della classe Report sono:

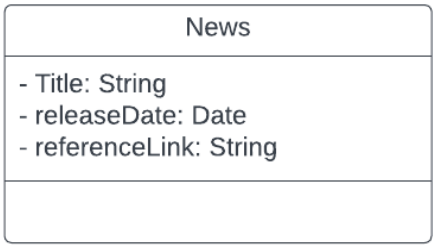


- **startDate/EndDate**: permette all'utente di scegliere attraverso la classe di supporto Date un periodo per la creazione del Report
- **budgetIncluded/TransactionList**: permette all'utente di creare una lista per le classi Budget e Transaction per i calcoli del Report

I metodi della classe Report sono:

- **requestTransactionForBudget**: attraverso una lettura del DB permette la creazione di una lista delle transazioni per il budget selezionato.
- **generateChart:** Genera il grafico che rappresenta i movimenti relativi al Budget
- **SumExit/SumEnter**: questo metodo esegue la somma delle transazioni in entrata e uscita per vedere un totale.
- **DiffInOut**: questo metodo esegue la differenza di entrate e uscite.

Classe News

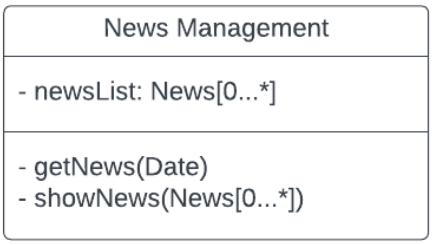


Gli attributi della classe News sono:

- **Title**
- **ReleaseDate**
- **referenceLink**

questi attributi servono per l'identificazione di ogni News

Classe News Management



Gli attributi della classe News sono:

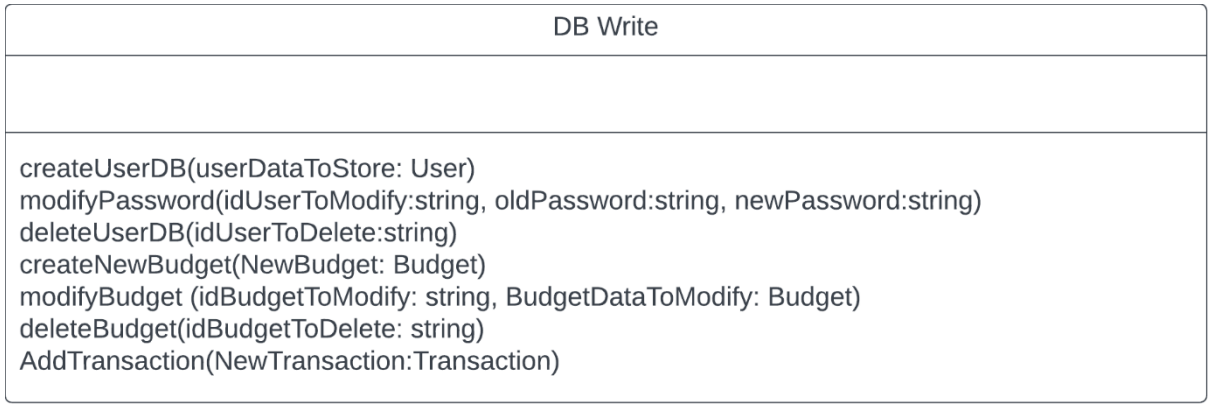
- **NewsList**: questo attributo contiene un insieme delle istanze della classe News

I metodi della classe News sono:

- **getNews**: permette attraverso l'inserimento di una data attraverso la classe di supporto Date di richiedere le News che hanno la stessa RelaseDate
- **showNews**: questo metodo permette la visualizzazione della lista delle News

Classe DBRead / DBWrite

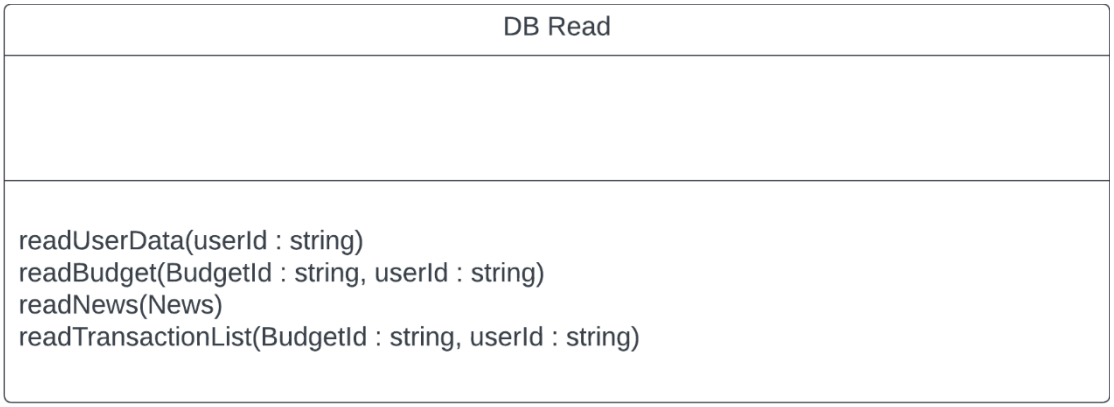
Le classi DBRead e DBWrite caratterizzano i componenti dell'applicazione che hanno un compito di interagire con una Base di Dati esterna all'applicazione. Implementa le funzionalità accessibili da diverse parti dell'applicazione relative alla scrittura/lettura delle informazioni relative all'utente manipolando su dati della classe User, alla scrittura/lettura delle informazioni relative ai budget e alle transazioni rappresentati dalle classi Budget e Recurrent Transaction /Non Recurrent Transaction.



I metodi implementati dalla classe DB Write quindi sono:

- **createUserDB(userDataToStore: User)**: vengono passati i dati forniti durante la registrazione del nuovo utente per la creazione di un nuovo record all'interno di DB.
- **modifyPassword(idUserToModify: String, oldPassword: String, newPassword: String)**: se l'utente autenticato caratterizzato dall'istanza della classe Logged-in User con il relativo id richiede la modifica della password in uso attraverso la sezione Account Management, i dati forniti vengono aggiornati nel DB
- **deleteUserDB(idUserToDelete: string):** se l'utente autenticato caratterizzato dall'istanza della classe Logged-in User con il relativo id richiede l'eliminazione dell'account, vengono eliminati i relativi dati nel DB
- **createNewBudget(NewBudget: Budget):** vengono passati e salvati nel DB i dati forniti durante la creazione di una nuova istanza della classe Budget
- **modifyBudget(idBudgetToModify: string, userId: string, BudgetDataToModify: Budget):** vengono passati attraverso BudgetDataToModify i campi aggiornati relativi all'idBudgetToModify e modificati all'interno del record nel DB se l'utente autenticato che richiede l'operazione è il proprietario del Budget.
- **deleteBudget(idBudgetToDelete: string, userId: string):** vengono eliminati I dati relativi al Budget identificato da idBudgetToDelete se l'utente autenticato che richiede l'operazione è il proprietario del Budget.
- **AddTransaction(NewTransaction: Recurrent Transaction/Non Recurrent Transaction)**: vengono passati e salvati nel DB I dati forniti durante la creazione di una nuova istanza della classe Transaction (Recurrent oppure Non Recurrent).

In modo simile, il componente, il cui compito è il prelievo dei dati salvati dalla Base di Dati è rappresentato dalla classe DB Read.



I metodi implementati dalla classe DB Read sono:

- **readUserData(userId: string)**: Legge dal DB i dati relativi all'utente con identificativo userId.
- **readBudget(BudgetId: string, userId: string):** Legge dal DB I dati relativi al budget con identificativo BudgetId che è associato all'utente con identificativo userId.
- **readNews(News)**: Legge dal DB una notizia da inviare al News Management
- **readTransactionList(BudgetId: string, userId: string):** Legge le transazioni associato al budget con identificativo BudgetId associato a sua volta all'utente con identificato userId.

3. Vincoli in Object Constraint Language

Nella seguenze sezione vengono formalmente descritti i vincoli applicati ad alcune delle classi nell'architettura dell'applicazione utilizzando il linguaggio Object Constraint Language. Tale linguaggio è complementare alla rappresentazione tramite diagrammi delle classi in UML, perché permette di introdurre e definire le regole e vincoli che non possono essere espressi tramite solamente diagrammi. I vincoli sono comunque essenziali per garantire la coerenza nella logica del funzionamento del sistema e garantire i comportamenti previsti dallo sviluppatore.

I vincoli in OCL si applicano agli specifici attributi sotto forma di invarianti, che prescrivono che il valore dell'attributo rispetti tale condizione per tutta la durata della sua vita. Un'altra tipologia di vincolo si riferisce ai metodi delle classi. Nello specifico, le pre-condizioni garantiscono che un metodo specifico soddisfi un prerequisito definito nella pre-condizione prima di poter essere eseguito. Le post-condizioni garantiscono che i requisiti definiti siano soddisfatti subito dopo l'esecuzione di un metodo.

Classe User

Utente non può non avere nome e cognome, ma questi non possono essere più lunghi di 30 caratteri per la questione di risparmio dello spazio durante la fase di archiviazione in DB. In più il campo dell'indirizzo email non può essere vuoto perché rappresenta un'informazione essenziale per il funzionamento corretto dell'applicazione.

---

context User

inv : self.name != NULL AND self.name->length() <= 30

inv : self.surname != NULL AND self.surname->length() <= 30

inv : self.email != NULL

Il metodo login non può essere eseguito se l'utente non risulta essere verificato, ovvero non ha passato la verifica della validita dell'account durante la fase di registrazione.

---

context User::login(email: string, password: string)

pre : self.verifyValidity(email: string, password: string) = TRUE AND self.verified = TRUE

Classe Registration

L'account può passare la conferma d'identità solo se passa ogni singola verifica precedente che incude verifica dell'unicità della mail registrata, la verifica della sicurezza della password e la conferma che sia il vero proprietario della mail.

---

context Registration

inv : newUser.email = User.email

context Registration::identityConfirmation(newUser: User)

pre : self.verifyEmailTaken(newUser.email: string) = FALSE AND
self.verifyStrongPassword(newUser.password: string) = TRUE AND
self.verifyEmail(newUser.email: string) = TRUE

La conferma della registrazione viene propagata soltanto se le verifiche necessarie sono andate a buon fine.

---

context Registration::confirmRegistration(newUser: User)

pre : self.identityConfirmation(newUser: User) = TRUE

post : self.operationSuccess = TRUE

Classe Registration With Google

I vincoli specificati permettono di evitare errori e garantire la sequenza temporale corretta per l'esecuzione di metodi della classe.

---

context RegisterAndLoginWithGoogle::verifyUserToken()

pre : self.getUserToken() != NULL

context RegisterAndLoginWithGoogle::getUserData()

pre : self.verifyUserToken() = TRUE

post : self.operationSuccess = TRUE

Classe Login

Il processo di login non può essere avviato se l'utente non è verificato.

---

context Login::confirmLogin(emailToCheck: string, passwordToCheck: string)

pre : User.verifyValidity(email: string, password: string) = TRUE

Classe Logged-In User

Seguenti vincoli garantiscono la corretta propagazione di informazioni e il funzionamento del logout solo per utenti autenticati che dovranno riautenticarsi dopo la sua esecuzione.

---

context LoggedInUser

inv: self.email = User.email

context LoggedInUser::logout()

pre : Login.confirmLogin(emailToCheck: string, passwordToCheck: string) = TRUE

post : Login.confirmLogin(emailToCheck: string, passwordToCheck: string) = FALSE

Classe Account Management

Seguente vincolo garantisce la corretta propagazione di informazioni.

---

context AccountManagement

inv : self.id = LoggedInUser.id AND self.email = LoggedInUser.email

---

La modifica della password può avvenire soltanto se viene fornita anche la password attualmente in uso per ragioni di sicurezza.

---

context AccountManagement::modifyPassword(id: string, password: string, newPassword: string)

pre: password = LoggedInUser.password

Classe Budget Management

Garantisce la visualizzazione di ogni Budget di cui il richiedente è il proprietario.

---

context BudgetManagement

inv : self.budgetsAvailable->forAll(Budget | Budget.userID = LoggedInUser.id)

Anche se l'utente, durante il corretto funzionamento dell'applicazione, non potrà accedere ai Budget degli altri utenti, la possibilità di eliminare un Budget solo se si è il suo proprietario funge da un ulteriore strato di sicurezza.

---

context BudgetManagement::removeBudget(Budget.id)

pre : Budget.userId = LoggedInUser.id

Classe Budget

Un Budget non può non avere nome, ma questo non può essere più lungo di 30 caratteri per la questione di risparmio dello spazio durante la fase di archiviazione in DB. Il campo delle note è opzionale ma non può superare 200 caratteri per lo stesso motivo.

Inoltre, un Budget deve tenere conto soltanto delle transazioni che sono associate a quel Budget tramite un loro attributo budgetId.

---

context Budget

inv : self.name != NULL AND self.name->length() <= 30

inv : self.note->length() <= 200

inv : self.transactionsAvailable->forAll(Transaction\| Transaction.budgetID = Budget.id)

Una nuova data di scadenza del Budget non può essere un giorno precedente alla data odierna.

---

context Budget::modifyExpiration()

post : self.expirationDate >= Date.getCurrentDate()

Il nome del Budget dopo la modifica non può risultare nullo od essere superiore a 30 caratteri.

---

context Budget::modifyName()

post : self.name != NULL AND self.name->length() <= 30

La nota del Budget dopo la modifica non può risulate superiore a 200 caratteri.

---

context Budget::modifyNote()

post : self.note->length() <= 200

Classe Non Recurrent Transaction

Una transazione non può non avere nome, ma questo non può essere più lungo di 30 caratteri per la questione di risparmio dello spazio durante la fase di archiviazione in DB. Il suo tipo, importo e categoria non possono essere dei campi vuoti. La data di inserimento non può essere una data successiva alla data di scadenza del budget. Il campo delle note è opzionale ma non può superare 200 caratteri.

---

context NonRecurrentTransaction

inv : self.name != NULL AND self.name->length() <= 30

inv : self.type != NULL

inv : self.balanceDifference != NULL

inv : self.category != NULL

inv : self.insertionDate < Budget.expirationDate

inv : self.note->length() <= 200

La data di inserimento della transazione dopo la modifica non può essere successiva alla data di scadenza del budget.

---

context NonRecurrentTransaction::modifyInsertionDate()

post : self.insertionDate < Budget.expirationDate

Classe Recurrent Transaction

Per quanto riguarda le transazioni ricorrenti, hanno gli stessi vincoli sugli attributi come le transazioni non ricorrenti. In più però, il periodo di ricorrenza non può essere inferiore a 1 giorno e la data d'inizio delle ricorrenze non può essere impostata ad una data successiva alla data di scadenza del Budget.

---

context RecurrentTransaction

inv : self.name != NULL AND self.name->length() <= 30

inv : self.type != NULL

inv : self.balanceDifference != NULL

inv : self.category != NULL

inv : self.recurrencePeriod >= 1

inv : self.startDate < Budget.expirationDate

Tale data deve rispettare il vincolo anche dopo la sua modifica.

---

context RecurrentTransaction::modifyStartDate()

post : self.startDate < Budget.expirationDate

Il periodo di ricorrenza deve rispettare il vincolo descritto precedentemente anche dopo la sua modifica.

---

context RecurrentTransaction::modifyRecurrencePeriod()

post : self.recurrencePeriod > 0

Classe Report

Il periodo per la generazione del report deve essere caratterizzato due date non nulle di cui quella della fine del periodo deve essere successiva alla data d'inizio. I Budget inclusi devono tutti avere il richiedente come proprietario e per ogni Budget incluso devono essere prelevate tutte le transazioni associate.

---

context Report

inv : self.startDate != NULL AND self.endDate !=NULL AND self.startDate <= self.endDate

inv : self.budgetIncluded->forAll(Budget | Budget.userId = LoggedInUser.id)

inv : self.transactionList->forAll(Transaction | self.budgetIncluded->exists(Budget | Budget.id = Transaction.budgetId))

Classe DB Write

Prima di essere creato un record relativo ad un nuovo utente all'interno del DB, l'operazione di registrazione deve essere conclusa con successo.

---

context DBWrite::createUserDB(userDataToStore: User)

pre : Registration.operationSuccess = TRUE OR RegisterAndLoginWithGoogle.operationSuccess = TRUE

Classe Amount€

Un vincolo per le invarianti logiche di un importo.

---

context AmountE

inv : intPart >= 0

inv : cent >= 0 AND cent <= 99

Classe Date

Un vincolo per le invarianti logiche di una data.

---

context Date

inv: year >= 1970 AND year <= 2050

inv: month >= 1 AND month <= 12

inv: day >= 1 AND day <= 31

---

4. Diagramma delle Classi completo

Qui rappresentato è il diagramma completo delle classi con tutte le interazioni rappresentate dai collegamenti e ogni vincolo in linguaggio OCL descritto precedentemente.