16/12/2018

ODD ShareErasmus

Share your Erasmus experience

Riferimento	
Versione	1.0
Data	16/12/2018
Destinatario	Prof.ssa F. Ferrucci
Proposto da	Federico Vitale, Francesco Vicidomini
Approvato da	

A.A. 2018/2019

Revision History

Data	Versione	Descrizione	Autori
07/12/2018	1.0	Prima stesura, introduzione	Davide Bottiglieri Francesco Breve Rosaria Iorio
08/12/2018	1.0	Design pattern, packages	Davide Bottiglieri Francesco Breve Rosaria Iorio
10/12/2018	1.0	Class interfaces, Glossario	Francesco Breve Rosaria Iorio
12/12/2018	1.0	Revisione	Davide Bottiglieri

Team composition

Ruolo	Nome	Posizione	Contatti
Top Manager	Filomena Ferrucci	Rappresentante del cliente	f.ferrucci@unisa.it
Project Manager	Francesco Vicidomini	Project Manager	f.vicidomini14@studenti.unisa.it
Project Manager	Federico Vitale	Project Manager	f.vitale40@studenti.unisa.it
Team Member	Alfonso Ruggiero		a.ruggiero114@studenti.unisa.it
Team Member	Davide Bottiglieri		d.bottiglieri4@studenti.unisa.it
Team Member	Francesco Breve		f.breve@studenti.unisa.it
Team Member	Giuseppe Cavaliere		g.cavaliere10@studenti.unisa.it
Team Member	Paolo Cantarella		p.cantarella1@studenti.unisa.it
Team Member	Rosaria Iorio		r.iorio11@studenti.unisa.it
Team Member	Silvio Corso		s.corso1@studenti.unisa.it
Team Member	Vincenzo Sabato		v.sabato1@studenti.unisa.it

1. Introduzione

1.1 Obeject Design Trade-offs

Dopo la stesura del documento di Requirements Analysis e il documento di System Design risulta necessario concentrare l'attenzione sugli aspetti implementativi. L'Object Design document ha quindi come obiettivo quello di produrre un modello coerente e preciso con tutte le informazioni accumulate collezionate durante le fasi precedenti. Il documento prevede quindi di dare: una panoramica del sistema, tutte le interfacce delle classi, le operazioni supportate, i tipi dei dati, i parametri delle procedure, i signatures dei sottosistemi definiti nel documento di System Design e linee guida e trade-offs.

Prestazioni vs Costi:

Nonostante il budget disponibile sia allocato quasi interamente in risorse umane tramite l'utilizzo degli account studenti è stato possibile ottenere accesso alle macchine virtuali di Amazon che forniscono capacità di elaborazione sicura e scalabile nel cloud. Tutta via attraverso queste macchine stiamo rinunciando all'immediatezza nei tempi di risposta poiché le macchine di Amazon si trovano in Oregon nella East Coast Americana.

Interfaccia vs Usabilità:

Questa piattaforma nasce per rendere più semplice ed intuitive varie operazioni collegate alla tematica Erasmus, in virtù di ciò verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale

Sicurezza vs Efficienza:

La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis Document, rappresenta uno degli aspetti principali del sistema; però causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su email e password.

1.2 Linee guida per la documentazione di interfacce

Gli sviluppatori dovranno seguire precise linee guida per la stesura del codice, le quali saranno dettate dallo stile "Google Java".

Naming Convention:

- Descrittivi;
- Pronunciabili;
- Di lunghezza media;
- Non abbreviati;
- Utilizzando solo caratteri consentiti.

Variabili:

- I nomi delle variabili dovranno iniziare con lettera maiuscola, i nomi delle parole successive

con lettera maiuscola evitando l'utilizzo del'underscore "_"
- Ogni variabile sarà dichiarata su un'unica riga
- L'utilizzo dell'carattere underscore "_" è da limtirasi alla dichiarazione di costanti

Metodi:

- I nomi dei metodi seguiranno la "Camel Notation", avendo come prima lettera iniziale quella minuscola e le parole successive lettera maiuscola.
- I metodi dovranno essere raggruppati in base alle loro funzionalità.
- Ogni metodo dovrà avere una descrizione delle sue funzionalità, parametri di input ed output

Classi Node.js

- I nomi delle classi dovranno iniziare con lettera maiuscola e così come le parole successive all'interno del nome.e le parole successive all'interno del nome.
- I nomi delle classi dovranno essere esplicativi e senza abbreviazioni
- Ogni classe dovrà avere una breve lista dei metodi che implementa.

Packages:

- I nomi dei packages dovranno iniziare con lettera minuscola
- Non saranno ammessi caratteri speciali

1.3 Definizioni acronimi e abbreviazioni

1.3.1 Definizioni

- **Design Pattern**: descrizione o modello logico da applicare per la risoluzione di un problema incontrato durante le fasi di progettazione e sviluppo del software.
- **Package**: collezione di classi e interfacce correlate.
- **Package model**: pacchetto che include i model del progetto, ossia la componente del modello MVC che si occupa dell'interazione tra l'applicazione e il database.
- **Package routes**: pacchetto che include i controller del progetto,ossia la componente del modello MVC che si occupata di elaborare le richieste di un utente e di comunicare con il model.
- **Package docs**: pacchetto che include le view del progetto ,ossia la componente del modello MVC scritto in linguaggio HTML e visualizzato nella pagina web dall'utente che permette all'utente di interagire col sistema.
- **MVC**: modello architetturale che si basa sull'utilizzo di 3 componenti fondamentali (model,view e controller) per lo sviluppo di web application.
- **Class diagram**: Tipo di diagramma che descrive la struttura di un sistema mostrando le sue classi, gli attributi di tali classi, le operazioni (o metodi) e le relazioni tra gli oggetti.

1.3.2 Acronimi e abbreviazioni

- **SE_ODD_Vers.1.0:** Utilizzata per indicare l'Object Design Document.
- **SE_RAD_Vers.1.2:** Utilizzata per indicare il Requirement Analysis Document.
- **SE_SDD_Vers.1.1:** Utilizzata per indicare il System Design Document.
- **SE_SOW_Vers.1.0:** Utilizzata per indicare lo Statement of Work.
- **DAO:** Utilizzata per indicare il pattern Data Access Object.

1.4 Riferimenti

- Kathy Schwalbe, "Information Technology Project Management", International Edition 7E, Cengage Learning, 2014;
- Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering Using UML, Patterns and Java", Third Ed., Pearson, 2010;
- Sommerville, "Software Engineering", Addison Wesley;
- PMBOK ® Guide and Software Extention to the PMBOK® Guide, Fifth Ed., Project Management Institute, 2013

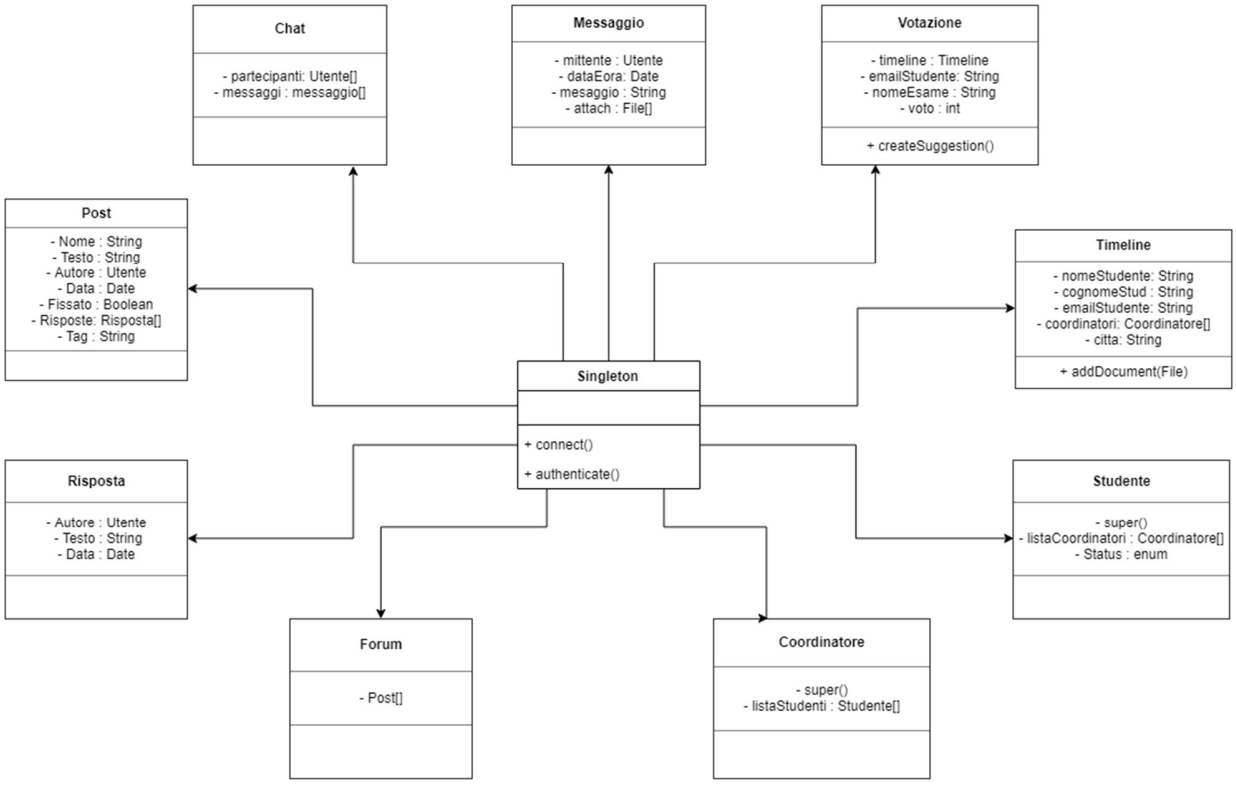
Documentazione di Progetto:

- SE_RAD_Vers.1.2;
- SE_SDD_Vers.1.1;
- SE_SOW_Vers.1.0;

2. Design Pattern

2.1 Design pattern Singleton

Il singleton è un design pattern che ha lo scopo di garantire che di una determinata classe ne venga creata una ed una sola istanza e di fornire un punto di accesso globale alla classe. Implementiamo il singleton pattern tramite l'utilizzo di Sequelize.



Il design pattern Singleton ha come scopo:

- Avere un accesso controllato all'unica istanza della classe
- Ridurre il numero di oggetti condivisi
- Centralizzare informazioni e comportamenti in un'unica entità condivisa dagli utilizzatori

Il principale vantaggio offerto è:

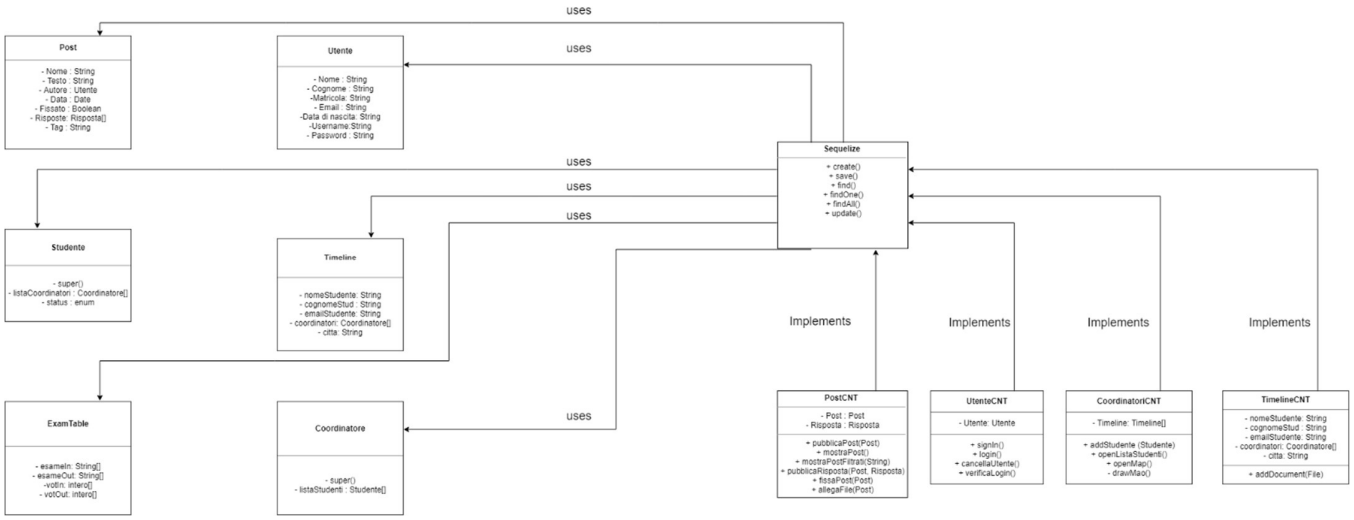
- Mutua esclusione

Utilizzo

Il design pattern verrà utilizzato per gestire l'accesso alla classe Logger. Qualsiasi classe che intende utilizzare i metodi della classe Logger dovrà farlo attraverso l'utilizzo dell'unica istanza della classe presente all'interno del sistema.

2.2 Design pattern DAO

DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare usata principalmente in applicazioni web.



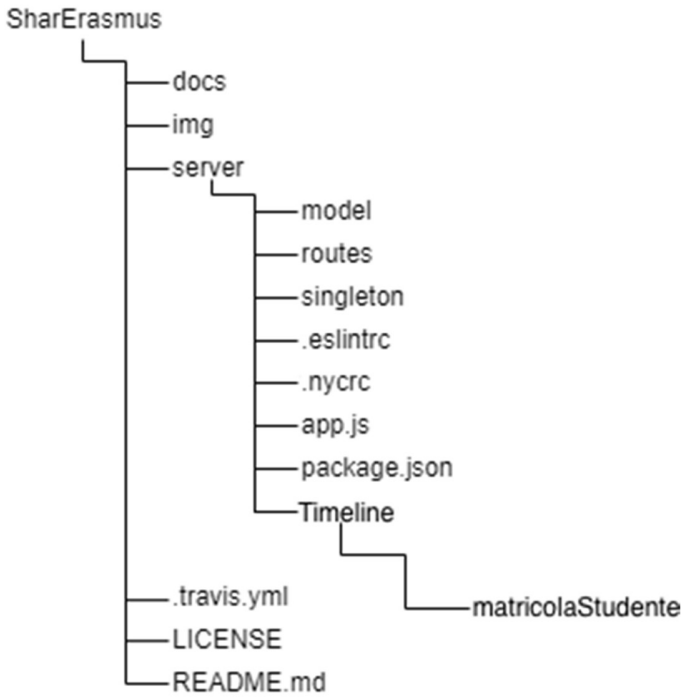
2.3 Design pattern Observer

Il pattern Observer (noto anche col nome Publish-Subscribe) permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente. L'Observer nasce dall'esigenza di mantenere un alto livello di consistenza fra classi correlate, senza produrre situazioni di forte dipendenza e di accoppiamento elevato.

Utilizzo

Il pattern Observer verrà utilizzato per gestire il sistema di notifiche, ovvero ogniqualvolta un utente pubblica un post con dei tag utilizzati già precedentemente da altri utenti, la classe PostControl notificherà tutti i client.

3. Package



(togliere text e bin, aggiungere Timeline->NomeStudente...)

Cartelle

- Docs: Contiene tutte le view
- Img: Contiente tutte le immagini del repsitory
- Model: Contiene tutti I model
- Routes: Contiene tutti I controller
- Singleton: Contiene il singleton che consente la connessione al database
- Timeline: Questa cartella conserva i documenti caricati sulle timeline di ogni studente. Ogni studente ha la propria sottocartella indentificata dalla matricola.

Dipendenze:

- Controller dipendono dai model
- Tutte le classi che si interfacciano con il database dipendono dal singleton.js

3.1 Package model

Classe	Descrizione
Utente.js	Classe che rappresenta un utente non registrato
Studente.js	Classe che rappresenta un utente registrato come studente
Coordinatore.js	Classe che rappresenta un utente registrato e certificato come coordinatore di un'esperienza Erasmus
Post.js	Classe che rappresenta un post scritto sul forum
Timeline.js	Classe che contiene le informazioni Erasmus degli studenti
ExamTable.js	Classe che rappresenta la tabella esami di uno studente contenuta nella timeline

Autore: Francesco Breve

3.2 Package routes

Classe	Descrizione
UtenteCNT.js	Controller per l'oggetto utente, si preoccuperà delle operazioni di sign in, login, cancellazione utente, la verifica di login e il rating.
PostCNT.js	Controller per la gestione del post, ha i metodi per la pubblicazione del post, la sua visualizzazione, la visualizzazione per tag, la risposta, la possibilità di fissare il post e di allegarci un file nel caso sia un avviso di un coordinatore.
CoordinatoriCNT.js	Controller per la gestione coordinatori, ha come metodi l'aggiunta di uno studente e l'apertura della lista di tutti gli studenti.
TimelineCNT.js	Controller per l'oggetto timeline, verrà utilizzata per l'inserimento di uno studente, il monitoraggio dello stato di quest'ultimo, l'aggiunta di documenti, l'inserimento degli esami conseguiti e la visualizzazione della tabella ECTS.

Autore: Davide Bottiglieri

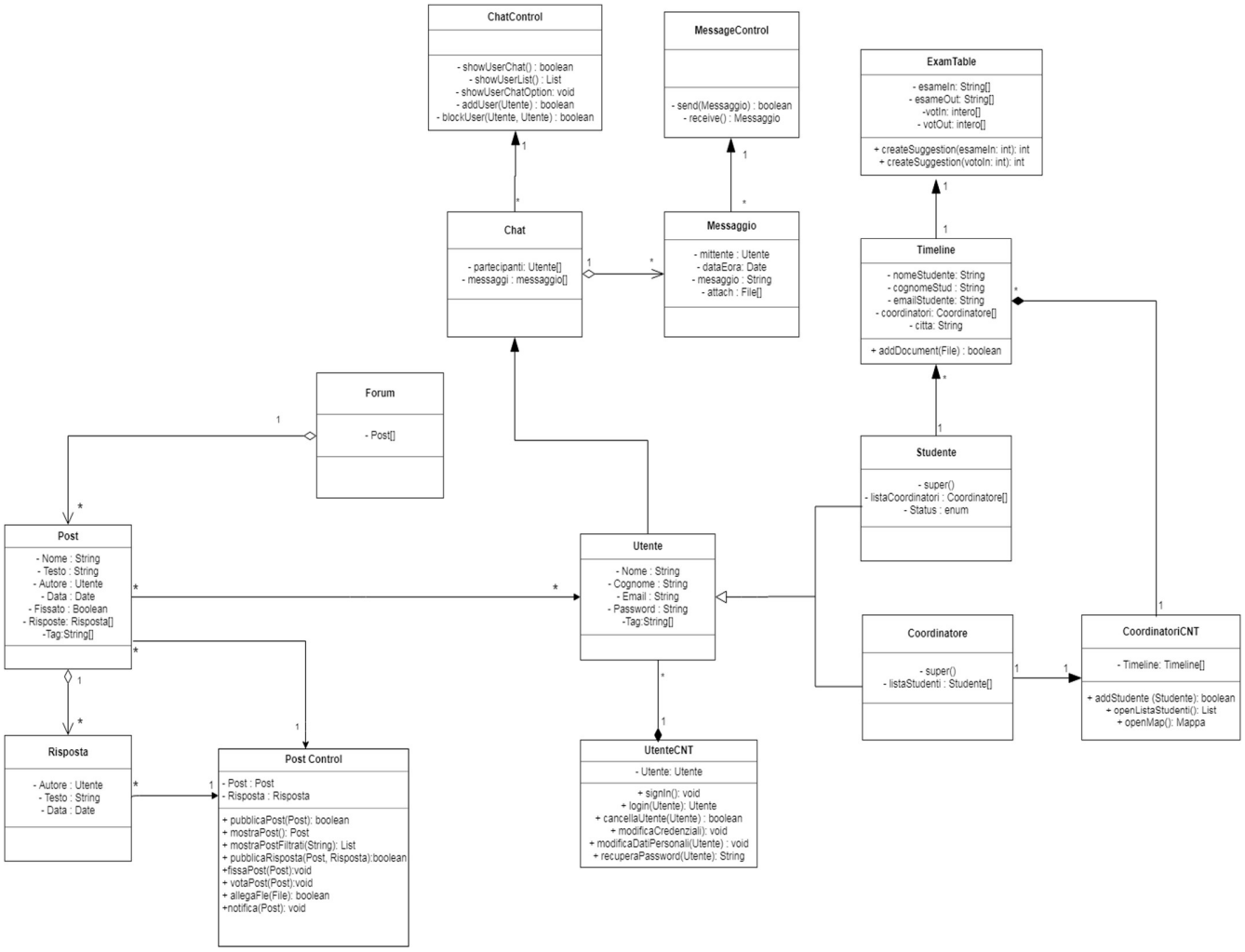
3.3 Package docs

Pagina	Descrizione
accessoNegato.html	Pagina mostrata quando non si possiedono i requisiti per visualizzare una determinata pagina o eseguire una determinata azione.
aggiungiStudente.html	Sezione che permette al Coordinatore di aggiungere un nuovo studente alla sua lista studenti.
bachecaAvvisi.html	Pagina che mostra tutti gli avvisi pubblicati dai coordinatori.
conferma.html	Pagina che richiede conferma all'Utente prima di eseguire un'operazione.
errore.html	Pagina che notifica un errore all'Utente in caso di insuccesso di un'operazione.
footer.html	Sezione footer comune a tutte le pagine ed include la sezione Chat.
forum.html	Pagina che mostra tutti i post pubblicati dagli utenti e comprende la funzionalità di ricerca per tag e la funzionalità di rispondere ad un post.
header.html	Sezione header comune a tutte le pagine.
homeCoordinatori.html	Pagina principale del coordinatore da cui si può accedere alle varie funzionalità del sistema.
homeStudente.html	Pagina principale dello studente da cui si può accedere alle varie funzionalità del sistema.
index.html	Pagina principale del sistema da cui si può effettuare login e registrazione.
listaStudenti.html	Sezione che mostra ad un coordinatore la lista degli studenti e comprende la funzionalità di ricerca studente.
mappa.html	Sezione che mostra una mappa europea con la quantità di studenti in Erasmus per ogni località

menu.html	Sezione menu.
modificaCredenziali.html	Sezione in cui si può modificare la propria password o il proprio username.
modificaProfilo.html	Sezione che permette di modificare le informazioni del proprio profilo.
notifiche.html	Sezione che mostra le notifiche ricevute.
profiloPersonale.html	Sezione che mostra informazioni sul profilo dell'utente.
recuperaPassword.html	Sezione in cui si può effettuare il recupero della password.
registrazione.html	Sezione in cui l'utente può effettuare la registrazione compilando i campi richiesti.
successo.html	Pagina che informa l'Utente del successo di un'operazione
tabellaEsami.html	Sezione che mostra la tabella ECTS
timeline.html	Pagina che mostra la timeline di uno studente.
uploadFile.html	Sezione che permette di caricare un documento.

Autore: Rosaria Iorio.

4. Class interfaces



Nome file	Utente.js
Descrizione	Questa classe rappresenta le informazioni relative ad un utente del sistema.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	Studente.js
Descrizione	Questa classe rappresenta le informazioni relative ad uno studente del sistema.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	Coordinatore.js
Descrizione	Questa classe rappresenta le informazioni relative ad un coordinatore del sistema.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	**UtenteCNT.js**
Descrizione	Consente di gestire le richieste di autenticazione degli utenti attraverso la procedura di login, le richieste di chiusura della sessione attraverso il logout, di effettuare la registrazione, la rimozione di un utente, la modifica delle credenziali, la modifica dei dati e recupero password.
Pre-condizione	post ('user/login', **function**(req, res) **pre:** req!=null&&res!=null post ('user/logout', **function**(req, res) **pre:** req!=null&&res!=null session.getAttribute('login')==true post ('user/signin,**function**(req, res) **pre:** req!=null&&res!=null post ('user/deleteAccount',**function**(req, res) **pre:** req!=null&& res!=null&& session.getAttribute('login')==true post ('user/modificaCredenziali',**function**(req, res) **pre:** req!=null&&res!=null && session.getAttribute('login')==true post ('user/modificaDatiPersonali,**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('user/recuperoPassword,**function**(req, res) **pre:** req!=null&&res!=null
Post-condizione	-
Invarianti	-

Nome file	Post.js
Descrizione	Definisce la struttura di un post pubblicato nel Forum
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	PostControl.js
Descrizione	Questa classe gestisce le operazioni relative ad un post
Pre-condizione	post ('forum/pubblicaPost',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('forum/mostraPost',**function**(req, res) **pre:** req!=null&&res!=null post ('forum/mostraPostFiltrati',**function**(req, res) **pre:** req!=null&&res!=null post ('forum/pubblicaRisposta',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('forum/fissaPost',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true session.getAttribute('certificato')==true Post ('forum/rating',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post('forum/notifica',**function**(req,res) **pre**: req!=null&&res!=null&& session.getAttribute('login')==true

Post-condizione	-
Invarianti	-

Nome file	**Risposta.js**
Descrizione	Definisce la struttura di una risposta relativa ad un post
Pre-condizione	**-**
Post-condizione	-
Invarianti	-

Nome file	**Forum.js**
Descrizione	Questa classe gestisce tutti i post pubblicati dagli utenti
Pre-condizione	**-**
Post-condizione	-
Invarianti	-

Nome file	**Chat.js**
Descrizione	Questa classe gestisce lo scambio di messaggi tra due utenti
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	ChatControl.js
Descrizione	Gestisce le operazioni relative ad una chat.
Pre-condizione	post 'chat/userChat',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('chat/userList',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('chat/addUser',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('chat/blockUser',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true
Post-condizione	-
Invarianti	-

Nome file	MessageControl.js
Descrizione	Consente di inviare o ricevere un messaggio in chat
Pre-condizione	post ('message/send',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true post ('message/receive',**function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true
Post-condizione	-
Invarianti	-

Nome file	Timeline.js
Descrizione	Definisce la struttura della timeline di uno studente e permette di caricare documenti
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	Votazione.js
Descrizione	Definisce un esame e il voto conseguito dall'utente
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome file	**CoordinatoriControl.js**
Descrizione	Consente ad un coordinatore di gestire l'operazione di aggiungere uno studente alla sua lista studenti e di visualizzare la mappa degli studenti in Erasmus.
Pre-condizione	post('coordinatore/addStudente', **function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true session.getAttribute('certificato')==true post('coordinatore/studentList', **function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true session.getAttribute('certificato')==true post('coordinatore/map', **function**(req, res) **pre:** req!=null&&res!=null&& session.getAttribute('login')==true session.getAttribute('certificato')==true
Post-condizione	-
Invarianti	-

5. Glossario

- **Utente:** rappresenta l'utilizzatore del sistema.
- **Studente:** rappresenta un utente autenticato che può effettuare diverse operazioni nel sistema ed è interessato all'attività di Erasmus.
- **Coordinatore (utente certificato):** rappresenta un utente autenticato che può effettuare diverse operazioni nel sistema ed interagisce con lo studente per gestire la sua attività di Erasmus.
- **Timeline:** rappresenta lo strumento mediante il quale un coordinatore si interfaccia con lo studente per gestire il progresso nell'Erasmus e la validazione dei suoi voti.
- **Chat:** 'conversazione' fra più interlocutori costituita da uno scambio di messaggi scritti che appaiono in tempo reale sul monitor di ciascun partecipante;
- **ECTS:** sistema di credito progettato per facilitare lo spostamento degli studenti tra diversi paesi;
- **Forum:** servizio che permette di inviare e leggere messaggi su un argomento specifico, che restano a disposizione per i commenti altrui; • Post: messaggio inviato all'interno del forum;
- **Profilo Utente**: pagina che mostra tutte le informazioni relative ad un utente;