Laurea Triennale in informatica - Università di Salerno
Corso di *Ingegneria del Software* - Prof.ssa F. Ferrucci

GUARDIAN FLOW

FEELING SAFE

Object Design Document

Guardian Flow

Riferimento	
Versione	0.6
Data	18/12/2023
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Intero team
Approvato da	Raffaele Mezza, Martina Mingione

Sommario

Revision History

Data	Versione	Descrizione	Autori
13/12/2023	0.1	Introduzione	Giuseppe Cerella Edmondo De Simone Mattia Guariglia
13/12/2023	0.2	Packages	Vincenzo Maiellaro Danilo Gisolfi Tommaso Nardi
14/12/2023	0.3	Class Interfaces	Tutto il team
15/12/2023	0.4	Class Diagram	Tutto il team
18/12/2023	0.5	Design Patterns	Tutto il team
18/12/2023	0.6	Glossario	Danilo Gisolfi
18/12/2023	0.6	Revisione	Vincenzo Maiellaro
18/12/2023	0.6	Approvazione	Martina Mingione Raffele Mezza

1. Introduzione

Dopo aver completato il documento di Requirements Analysis e il documento di System Design, in cui abbiamo delineato una panoramica del nostro sistema, identificato gli obiettivi senza entrare nei dettagli implementativi, ci apprestiamo a redigere il documento di Object Design.

L'obiettivo principale di questo documento è sviluppare un modello che integri in modo coerente e preciso tutte le funzionalità identificate nelle fasi precedenti. Inoltre, prevediamo di definire i packages, che saranno fondamentali per le decisioni implementative cruciali.

Questo documento fornirà un'articolata struttura di progettazione che permetterà l'integrazione delle funzionalità individuate, garantendo coerenza e precisione nel modello. I packages che saranno sviluppati rappresenteranno una base solida per le decisioni chiave nell'implementazione del sistema.

1.1 Object design goals

Portabilità

Il sistema mira a garantire elevata portabilità attraverso la realizzazione di componenti indipendenti dal contesto di esecuzione. Ciò consente al sistema di adattarsi agevolmente a differenti ambienti, piattaforme e dispositivi senza necessità di modifiche sostanziali, facilitando così l'integrazione e la distribuzione in contesti diversi.

Robustezza

Il sistema deve garantire la sua robustezza, reagendo in modo adeguato a situazioni impreviste attraverso opportuni controlli degli errori e una gestione efficace delle eccezioni.

Riusabilità

Nell'implementazione del sistema, il front-end sarà sviluppato mediante l'impiego di librerie mirate a promuovere la riusabilità, semplificando così il processo di creazione. Inoltre, per massimizzare l'efficienza nella gestione del codice, adotteremo i componenti appositamente creati dal nostro team.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida che abbiamo seguito per la progettazione delle interfacce nel nostro progetto utilizzano regole specifiche per i linguaggi che stiamo impiegando: Javascript e Python.

Abbiamo adottato convenzioni di stile e regole di codifica che sono state riferite a risorse ufficiali per garantire coerenza e chiarezza nel nostro lavoro:

- Per le convenzioni di stile in Nuxt.js, ci siamo basati sulla documentazione ufficiale di Nuxt.js che definisce le best practice e le convenzioni di codifica specifiche per questo framework.
  Link alla documentazione: Nuxt: The Intuitive Vue Framework · Nuxt
- Per quanto riguarda Python, abbiamo seguito le linee guida PEP 8, che rappresentano le convenzioni ufficiali di stile per il linguaggio Python, promuovendo la leggibilità del codice e la coerenza nel suo sviluppo.
  Link alla documentazione: PEP 8 – Style Guide for Python Code | peps.python.org

Queste risorse rappresentano le guide che abbiamo utilizzato per definire le regole e le convenzioni da seguire durante la progettazione delle interfacce nel contesto dei linguaggi Javascript e Python.

1.3 Definizioni, acronimi e abbreviazioni

1.3.1 Definizioni

- Design Goals: obiettivi di design progettati per il sistema proposto;
- LowerCamelCase: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura;
- Package: raggruppamento di classi, interfacce o file correlati;
- UpperCamelCase: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;

1.3.2 Acronimi e Abbreviazioni

- DG: Design Goals;

1.4 Riferimenti

RAD Requirement Analysis Document Guardian Flow V_1.6;

SDD System Design Document Guardian Flow V_1.3.

2. Packages

In questa sezione, mostriamo la suddivisione in package del nostro sistema, seguendo la struttura predefinita e impostata da Nuxt.js. Questa organizzazione dei pacchetti è stata definita in conformità al documento di System Design, che ha guidato le decisioni architetturali e il layout del nostro progetto in base alle esigenze stabilite in fase di progettazione.

La decisione di aderire a questa suddivisione è dettata dalla struttura stessa di Nuxt.js, che ci fornisce un modello predefinito di organizzazione dei file e dei pacchetti nel nostro progetto. Quest'approccio adotta la struttura standard fornita da Nuxt.js, consentendoci di aderire alle direttive di progettazione delineate nel documento di System Design mentre utilizziamo la convenzione di packaging definita dal framework Nuxt.js.

- **Guardian-flow**
    - **.nuxt**, contiene i file e le risorse generate durante il processo di compilazione e build del progetto
    - **assets**, è utilizzata per risorse che devono essere elaborate o manipolate da Nuxt.js durante la build dell'applicazione
    - **components**, contiene tutti i componenti che verranno utilizzati
    - **composables**, contiene tutti i composables utilizzati
    - **layouts**, contiene i file che definiscono i layout predefiniti per le diverse pagine del sito
    - **middleware**, contiene i vari middleware
    - **node_modules**, contiene tutti i moduli e le dipendenze esterne installate
    - **pages**, contiene tutte le pagine
        - **gestionale**, contiene tutte le pagine della dashboard

- **plugins**, contiene file JavaScript che vengono eseguiti prima dell'inizializzazione di Vue.js
- **public**, contiene le risorse statiche, come immagini, file CSS e font, che sono accessibili direttamente tramite il browser senza passare attraverso la gestione di webpack di Nuxt.js.
- **types**, contiene tutti i tipi utilizzati
- **server**, contiene la logica lato server
  - **api**, contiene file gestiscono le chiamate API

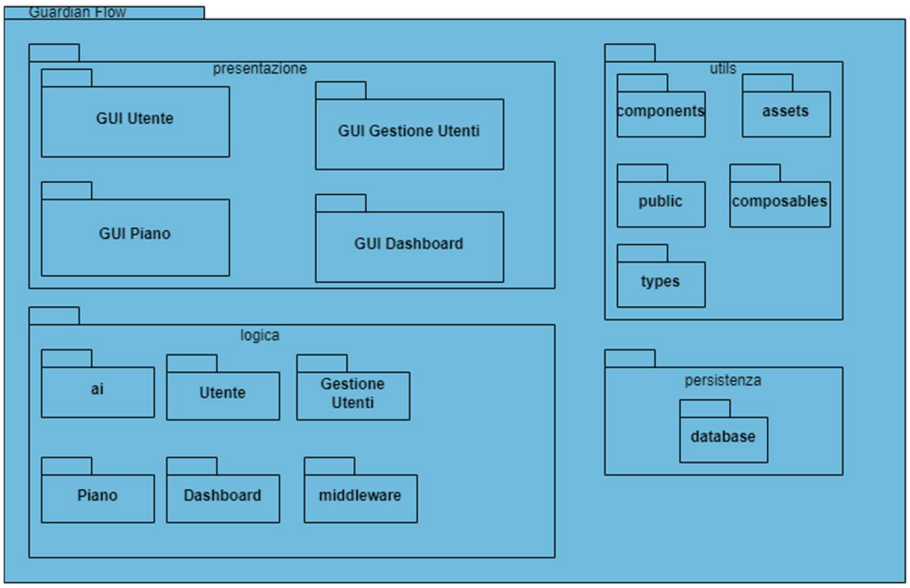
- **ai**, contiene tutti gli script in python

Package Guardian Flow

Nella presente sezione si mostra la struttura del package principale di Guardian Flow.

La struttura generale è stata ottenuta a partire da quattro principali scelte:

- Creare un package per il layer di presentazione chiamato "Presentazione";
- Creare un package per il layer di logica chiamato "Logica";
- Creare un package per il layer di persistenza chiamato "Persistenza";
- Creare un package chiamato "utils" in cui inserire eventuali file utili e usabili da più sottosistemi.



3. Interfaces

Abbiamo previsto le interfacce solamente per i file che implementano la logica di business nel sistema.

Nome file	RegistrazioneAzienda
Descrizione	Questo file permette di gestire le operazioni relative alla registrazione.
Metodi	+registrazione(Azienda azienda) : AziendaRegistrata
Invariante di classe	/

Nome Metodo	+registrazione(Azienda azienda) : AziendaRegistrata
Descrizione	Questa funzione consente di registrare una nuova azienda.
Pre-condizione	/
Post-condizione	context: RegistrazioneAzienda :: registrazione(Azienda azienda) post: db.insert(azienda) == true
Eccezioni	/

Nome file	LoginUtente
Descrizione	Questo file permette di gestire le operazioni relative al login di un utente.
Metodi	+login(String email, String password) : Utente +login2FA(String email, String password, int codice) : Utente +logout(Utente utente) : void
Invariante di classe	/

Nome Metodo	+login(String email, String password) : Utente
Descrizione	Questa funzione consente di effettuare l'accesso a un utente.
Pre-condizione	context: LoginUtente:: login(String email, String password) pre: email != null && password != null
Post-condizione	context: LoginUtente:: login(String email, String password) post: utente != null && utente istanceOf Amministratore || utente instanceOf Subordinato
Eccezioni	/

Nome Metodo	+login2FA(String email, String password, int codice) : Utente
Descrizione	Questa funzione consente di effettuare l'accesso con l'autenticazione a due fattori a un utente.
Pre-condizione	context: LoginUtente:: login2FA(String email, String password, int codice) pre: email != null && password != null && codice != null
Post-condizione	context: LoginUtente:: login2FA(String email, String password, int codice) post: utente != null && utente istanceOf Amministratore || utente instanceOf Subordinato
Eccezioni	/

Nome Metodo	+logout(Utente utente) : utente
Descrizione	Questa funzione consente di uscire dalla sessione a un utente.
Pre-condizione	context: LoginUtente :: logout() pre: login()!= true || login2FA()!=true
Post-condizione	context: LoginUtente :: logout() post: login2FA(utente) !=true|| login(utente) !=true
Eccezioni	/

Nome file	GestioneUtenti
Descrizione	Questo file permette di gestire le operazioni relative alla gestione di un utente.
Metodi	+creaUtente(String nome, String cognome, String email, String permessi) : Utente +rimuoviUtente(Utente utente) : void +modificaUtente(Utente utente) : void +getUtente() : Utente
Invariante di classe	/

Nome Metodo	+creaUtente(String nome, String cognome, String email, String permessi) : Utente
Descrizione	Con questa funzione è possibile creare un nuovo utente.
Pre-condizione	context: GestioneUtenti :: creaUtente(String nome, String cognome, String email, String permessi) pre: UtenteisAdmin() == true
Post-condizione	context: GestioneUtenti :: creaUtente(String nome, String cognome, String email, String permessi) post: db.insert(Utente) == true
Eccezioni	/

Nome Metodo	+modificaPassword(String password) : void
Descrizione	Con questo metodo è possibile modificare la password esistente.
Pre-condizione	context: Utente :: modificaPassword(String password) pre: utente != null
Post-condizione	context: Utente :: modificaPassword(String password) post: db.update(utente) == true
Eccezioni	PasswordNonValidaException

Nome Metodo	+rimuoviUtente() : Utente
Descrizione	Con questa funzione è possibile rimuovere un utente.
Pre-condizione	context: GestioneUtenti :: rimuoviUtente(Utente utente) pre: UtenteisAdmin() == true && utente() == true
Post-condizione	context: GestioneUtenti :: rimuoviUtente(Utente utente) post: db.drop(utente) == true
Eccezioni	/

Nome file	Utente
Descrizione	Questo file permette di gestire le operazioni relative al login di un utente.
Metodi	+modificaPassword(String password) : void +attiva2FA (String cellulare) : void +recuperaPassword () : String +getUtente() : Utente +checkisAdmin(Utente utente) : boolean
Invariante di classe	/

Nome Metodo	+attiva2FA(String cellulare) : void
Descrizione	Con questo metodo è possibile attivare l'autenticazione a 2 fattori.
Pre-condizione	context: Utente :: attiva2FA(String cellulare) pre: utente != null
Post-condizione	context: Utente :: attiva2FA(String cellulare) post: db.update(utente) == true
Eccezioni	/

Nome Metodo	+recuperaPassword () : String
Descrizione	Con questo metodo è possibile recuperare la propria password.
Pre-condizione	context: Utente :: recuperaPassword () pre: utente != null
Post-condizione	context: Utente :: recuperaPassword () post: db.select(utente) == true
Eccezioni	/

Nome Metodo	+ getUtente () : Utente
Descrizione	Con questo metodo è possibile recuperare un determinato utente.
Pre-condizione	context: Utente :: getUtente () pre: utente != null
Post-condizione	context: Utente :: getUtente () post: db.select(utente) == true
Eccezioni	/

Nome Metodo	+checkisAdmin() : boolean
Descrizione	Con questo metodo è possibile controllare i permessi dell'utente.
Pre-condizione	context: Utente :: checkisAdmin () pre: utente != null
Post-condizione	context: Utente :: checkisAdmin () post: db.select(utente) == true
Eccezioni	/

Nome file	Anomalia
Descrizione	Questo file permette di gestire le operazioni relative alle anomalie.
Metodi	+getAnomalie() : +segnalaFalsoPositivo(Anomalia anomalia) :
Invariante di classe	/

Nome Metodo	+getAnomalia() :
Descrizione	Con questa funzione è possibile visualizzare le anomalie.
Pre-condizione	context: Anomalia :: getAnomalia() pre: anomalia != null
Post-condizione	context: Anomalia :: getAnomalia () post: db.select(anomalia) == true
Eccezioni	/

Nome Metodo	+segnalaFalsoPositivo(Anomalia anomalia) : void
Descrizione	Con questa funzione è possibile segnalare un falso positivo.
Pre-condizione	context: Anomalia :: segnalaFalsoPositivo(Anomalia anomalia) pre: anomalia != null
Post-condizione	context: Anomalia :: segnalaFalsoPositivo(Anomalia anomalia) post: db.update(anomalia) == true
Eccezioni	/

4. Design Patterns

In questa sezione verranno mostrati i Design Patterns usati nello sviluppo del sistema.

Adapter

Il design pattern Adapter riveste un ruolo fondamentale nel contesto del nostro sistema. In particolare, il nostro sistema fa un ampio uso di componenti off-the-shelf, cioè soluzioni software già esistenti e pronte per l'utilizzo.
Questo approccio consente una facile integrazione di componenti software senza richiedere modifiche significative al codice esistente.
In questo modo, il pattern Adapter contribuisce a garantire una maggiore flessibilità e modularità nel nostro sistema, permettendo l'adozione agevole di soluzioni esterne senza compromettere la coerenza e la compatibilità. La strategia di utilizzo del pattern Adapter si configura come un elemento chiave nell'architettura complessiva, facilitando l'incorporazione di componenti eterogenei e contribuendo al successo e all'efficienza del sistema.

5.Glossario

D

Design Goals

Obiettivi di design progettati per il sistema proposto

E

Eccezione

Evento anomalo che interrompe il normale flusso di esecuzione del programma.

L

LowerCamelCase

La pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura.

O

Off-the-shelf

Un componente off-the-shelf si riferisce a un componente software o hardware che è disponibile sul mercato e può essere acquistato o utilizzato senza la necessità di svilupparlo internamente. Questi componenti sono già pronti per l'uso e sono stati progettati per soddisfare esigenze comuni o specifiche in diversi contesti.

P

Package

Raggruppamento di classi ed interfacce.

U

UpperCamelCase

La pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi.