Laurea Triennale in Informatica - Università di Salerno
Corso di *Ingegneria del Software* - Prof C. Gravino

Easy Pass

Object Design Document
Easy Pass

Riferimento	
Versione	1.0
Data	14/1/2022
Destinatario	Prof. C. Gravino
Presentato da	Montefusco Alberto Mulino Martina Rinaldi Viviana Spina Gennaro
Approvato da	

Sommario

Revision History

Data	Versione	Descrizione	Autori
13/12/2021	0.1	Prima stesura: divisione dei compiti	Alberto Montefusco Gennaro Spina Viviana Rinaldi Martina Mulino
13/12/2021	0.2	Aggiunta dei trade-off e delle linee guida per la documentazione delle interfacce	Alberto Montefusco Viviana Rinaldi
14/12/2021	0.3	Aggiunta dei package del progetto	Martina Mulino
15/12/2021	0.4	Aggiunta dei Design Pattern	Alberto Montefusco Gennaro Spina Viviana Rinaldi Martina Mulino
14/1/2022	1.0	Correzioni e Revisione finale	Alberto Montefusco Gennaro Spina Viviana Rinaldi Martina Mulino

Responsabilità

Artefatto	Autori
Object Design Trade-off	Viviana Rinaldi
Components off-the-shelf	Gennaro Spina

Linee guida per la documentazione delle interfacce	Alberto Montefusco
Package, Class Diagrams	Martina Mulino
Design Pattern	Alberto Montefusco, Viviana Rinaldi, Martina Mulino, Gennaro Spina
Interfacce delle classi Esito, Dipartimento e Formato	Alberto Montefusco
Interfacce delle classi PersonaleUnisa, Docente, DirettoreDiDipartimento	Martina Mulino
Interfacce delle classi Report e SessioneDiValidazione	Gennaro Spina
Interfacce delle classi ConnectionSingleton e Validator	Viviana Rinaldi

1. Introduzione

L'Object Design Document illustra i diversi dettagli legati alla fase implementativa del Sistema Easy Pass; in particolare, esso descrive gli object design goals, i trade-off di progettazione definiti dagli sviluppatori, le linee guida da seguire per le interfacce dei sottosistemi, i design pattern utilizzati, la decomposizione dei sottosistemi in packages e classi e, infine, la specifica delle interfacce delle classi.

1.1 Object Design Goals

Gli obiettivi di object design posti per il Sistema sono:

- **Astrazione**: le interfacce devono essere intuitive e di un alto livello, così da garantire un'implementazione corretta e comprensibile;
- **Modularità**: le unità del Sistema devono essere organizzate in moduli facilmente collegati;
- **Riusabilità**: il riuso del codice deve essere prioritario e verrà fornito attraverso l'ereditarietà e i design pattern.

1.2 Object Design Trade-off

Tempo di rilascio Vs funzionalità

Per rispettare le scadenze del Progetto potrebbe essere necessaria l'implementazione parziale di alcune delle funzionalità richieste.

Portabilità Vs Efficienza

Il Sistema è progettato per far in modo che Browser diversi e dispositivi aventi risoluzioni diverse (come smartphone, laptop o computer desktop) possano visualizzare correttamente le pagine web del sito Easy Pass sfruttando al meglio lo spazio del display. Questo implica che il livello di efficienza garantito non sia lo stesso per ogni dispositivo, poiché una tale adattabilità richiederebbe un carico maggiore da gestire.

Velocità Vs memoria

Per garantire tempi di risposta rapidi, si è preferito utilizzare query che risultano più veloci a discapito dello spazio che occupano in memoria; in particolar modo viene introdotta ulteriore ridondanza di dati.

Costruire Vs Comprare

Sebbene utilizzare software già realizzato da altri permetta, ad esempio, l'utilizzo di funzionalità già complete oppure una minore quantità di lavoro per gli sviluppatori, è stato deciso di realizzare la maggior parte del Sistema partendo da zero, utilizzando componenti esterne soltanto in alcuni casi (es. integrazione di librerie open source fornite dal Ministero della Salute per la validazione dei Green Pass). Il motivo per cui è stata presa questa decisione riguarda l'aumento dei costi e l'impegno necessario per integrare le componenti già realizzate con quelle costruite dagli sviluppatori.

Nella seguente tabella, il Design Goal in **grassetto** indica il design goal prioritario.

Trade-Off	
Tempi di rilascio	Funzionalità
Portabilità	Efficienza
Velocità	Memoria
Costruire	Comprare

1.3 Componenti off-the-shelf

Il Sistema utilizzerà i seguenti componenti off-the-shelf:

- **Apache Tomcat**: un Web Server con annesso application container per applicazioni scritte in Java;
- **Node.js**: un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript;
- **jsQR**: una libreria JavaScript per la lettura di codici QR da immagini;
- **dcc-utils**: un package NPM (Node Package Manager), fornito dal Ministero della Salute, il quale contiene una serie di tool per leggere e validare i Green Pass;

- **iTextPDF**: una libreria per la creazione e la gestione di file in formato pdf;
- **Qrcodegnerator**: libreria utilizzata per la creazione del QRcode della sessione di validazione. Il codice importato si trova nel package "io.nayuki.qrcodegen" nella cartella "java".

Per l'implementazione della sezione front-end sono stati utilizzati i seguenti framework:

- **bootstrap-5.1.3**: front-end open source toolkit;
- **Data-Tables-1.11.3**: plug-in per la libreria JQueryJavascript che aggiunge feature avanzate alle tabelle HTML.

1.4 Linee guida per la documentazione di Interfacce

Tali linee guida includono una lista di regole che gli sviluppatori devono seguire durante la progettazione delle interfacce. Di seguito sono è riportato un link alla convenzione usata per definire le linee guida su html:

- HTML: https://www.w3schools.com/html/html5_syntax.asp

Inoltre, il progetto Easy Pass è realizzato con l'IDE di sviluppo IntelliJ IDEA 2021.3 ed è strutturato nel seguente modo:

- Il progetto è suddiviso in tre package principali (Interface, Storage, ApplicationLogic) i quali contengono i rispettivi sub-package.
- Il nome di una classe deve rispettare il seguente formato: **NomeClasse**.
- Il nome di un metodo o di una variabile di istanza deve rispettare la notazione Camel Case.
- Un intero metodo, compreso di intestazione ed istruzioni, è preceduto e seguito da una riga vuota.
- I commenti, laddove necessari, avranno formato **// commento** se si estendono su una sola riga, altrimenti se un commento si estende su più righe presenta il formato **/* commento */**.
- Ogni classe e ogni metodo devono essere corredate da commenti che rispettano lo standard utilizzato da Javadoc per la produzione di documentazione in formato HTML.
- Il package "Storage" contiene dei sub-package in cui sono presenti tutte le classi che fanno riferimento ad entità persistenti (Bean, DAO e Mapper*****).

*I Mapper sono classi Java che riempiono un Bean prendendo i dati da un ResultSet restituito da una query sul database.

- Ogni classe che funge da Bean, all'interno del package "Storage", deve contenere almeno un costruttore e i metodi getter e setter.
- Il package "ApplicationLogic" contiene tutte le classi Servlet (@WebService) che si occupano della logica di business del Sistema e agisce da interlocutore tra le classi contenute nei sub-package del package Storage e Interface.
- Il sub-package "Ajax", che si trova nel package "ApplicationLogic", contiene le servlet responsabili della gestione delle richieste ajax effettuate dal client.
- Il package Interface contiene dei sub-package in cui sono organizzati tutti i file che si occupano dell'interfaccia utente (JSP, pagine HTML).
- Il package Interface contiene un sub-package "frontend-lib" che contiene le librerie utilizzate per la realizzazione delle JSP.
- I file CSS si trovano nel package WEB-INF del progetto e in particolare in un sub-package chiamato "css".
- I file JavaScript si trovano nel package WEB-INF del progetto e in particolare in un sub-package chiamato "js".

1.5 Definizioni, acronimi e abbreviazioni

In questa sezione descriveremo i termini che sono stati utilizzati all'interno del Documento stesso divisi in tre sezioni principali: definizioni, acronimi ed abbreviazioni.

1. **Definizioni**:
   - **Package**: raggruppamento di classi, interfacce, file correlati o altri package;
   - **Design pattern**: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
   - **Interfaccia**: insieme di signature delle operazioni offerte dalla classe;
   - **Camel Case**: 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 intermedi;

- **Javadoc**: sistema di documentazione offerto da Java, che viene generato sottoforma di pagina web in modo da rendere la documentazione accessibile e facilmente leggibile.
- **Maven**: strumento di gestione di progetti software ospitato da Apache Software Foundation.

2. **Acronimi**:
   - **SDD**: System Design Document
   - **RAD**: Requirements Analysis Document
   - **CRUD**: Create Read Update Delete

1.6 Riferimenti

Per stilare la presente documentazione, si è preso come riferimento le slide fornite dal Docente del corso di Ingegneria del Software, Carmine Gravino, inserite nella sezione "M4" della piattaforma di e-learning della facoltà di Informatica. Inoltre, è stato consultato il libro di testo "Object-Oriented Software Engineering Using UML, Patterns and Java: Third Edition, di Bernd Bruegge ed Allen H. Dutoit" e, infine, si è consultata la documentazione relativa al RAD e SDD.

2. Package

In questa sezione viene mostrata la suddivisione del Sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e sottolinea la struttura di directory standard definita da Maven.

2.1 Interface

Questo package contiene i seguenti sub-package e le seguenti classi:

- Package **StudenteGUI**
  - **InvioGP**: mostra allo Studente l'id della Sessione a cui è connesso e il form per l'inserimento del Green Pass;
  - **InvioEffettuato**: mostra un messaggio di notifica in cui si legge che l'invio del Green Pass è andato a buon fine.

- Package **DocenteGUI**
  - **AvvioSessione**: contiene il pulsante che permette di avviare una sessione di validazione e il form che permette di inserire il numero di studenti da validare;
  - **ElencoEsiti**: mostra la schermata con gli esiti delle validazioni effettuate;
  - **AnteprimaReport**: mostra un'anteprima del report relativo alla sessione di validazione appena effettuata.
- Package **DirettoreDiDipartimentoGUI**
  - **HomePage**: è la pagina che viene visualizzata subito dopo aver effettuato il login e che permette di accedere alle varie funzionalità previste per il Direttore di Dipartimento;
  - **GestioneReport**: pagina relativa alle funzionalità per la gestione dei report: contiene check-box, un elenco di report e una barra di ricerca con filtri;
  - **GestioneFormato**: pagina relativa alla funzionalità per la scelta del formato dei report: contiene i toggle necessari alla scelta.

- Package **AutenticazioneGUI**
  - **Autenticazione**: mostra il form di login usato da Docente e Direttore e permette anche di visualizzare il form dedicato alla registrazione di un nuovo Docente.

- Package **Errori**
  - **Error404**: viene visualizzata in caso di errore 404;
  - **Error401**: viene visualizzata in caso di errore 401;
  - **Error500**: viene visualizzata in caso di errore 500.

- Package **Partials**
  - **Head:** contiene meta-informazioni che riguardano la pagina in cui tale partial viene richiamato;
  - **Logout:** contiene il bottone che permette al docente di effettuare il logout;
  - Sub-package **Direttore:**
    - **AnteprimaReport**: mostra al Direttore l'anteprima del report selezionato dalla lista di report;
    - **Footer:** sezione informativa in fondo alla pagina;
    - **Header:** sezione di pagina in alto contenente il nome del Direttore loggato;
    - **Navbar:** area laterale contenente il menu di navigazione.

2.2 ApplicationLogic:

Questo package contiene i seguenti sub-package e le seguenti classi:

- Package **Ajax:**
  - **AjaxServlet**: servlet che gestisce le richieste AJAX, elabora i dati ricevuti e inoltra i dati richiesti.
- Package **Servlet**
  - **SessionController**: servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti la sessione di validazione;
  - **AccessController**: servlet che si occupa di svolgere e controllare le operazioni di autenticazione effettuate da Docenti e Direttore, quali login, logout e registrazione;
  - **ReportController**:servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti i report memorizzati.

- Package **Utils**
  - **ConnectionSingleton:** classe scritta utilizzando il design pattern del "Singleton", utilizzata per gestire la connessione al database;
  - **FileServlet:** servlet utilizzata per memorizzare un file all'interno del server web;

- **InvalidRequestException:** classe che gestisce l'eccezione solevata al momento di una richiesta HTTP non valida;
- **JsonSerializable:** interfaccia che fornisce un metodo per convertire un oggetto in tipo JSON;
- **ServletLogic:** servlet che contiene metodi di utility per le altre Servlet;
- **Alert:** classe che permette di gestire i messaggi di errore;

- Sub-Package **Validator**:
  - **Validator**: classe che contiene i metodi per validare i parametri di una request a seconda di parametri di validazione indicati;
  - **DocenteValidator**: classe che contiene i metodi per validare i parametri di una richiesta HTTP che corrispondono ai campi di un oggetto Docente, a seconda dei parametri di validazione indicati;
  - **StudenteValidator**: classe che contiene il metodo per validare l'invio del Green Pass;
  - **DirettoreValidator**: classe che contiene i metodi per validare i parametri di una richiesta HTTP che corrispondono ai campi di un oggetto Direttore, a seconda dei parametri di validazione indicati.

2.3 Storage

Questo package contiene i seguenti sub-package e le seguenti classi:

- Package **PersonaleUnisa**
  - **PersonaleUnisa**: classe astratta che modella una persona che lavora nell'università;

  - Sub-package **Docente**
    - **Docente**: classe che estende "PersonaleUnisa" e che modella un Docente;
    - **DocenteDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti Docente;
    - **DocenteMapper**: classe statica che fornisce i metodi necessari all'estrazione dei dati di un Docente da un ResultSet di ritorno di una query al Database;

  - Sub-package **Direttore**
    - **DirettoreDiDipartimento**: classe che estende "PersonaleUnisa" e che modella un Direttore di Dipartimento;

  - **DirettoreDiDipartimentoDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti DirettoreDiDipartimento;
  - **DirettoreDiDipartimentoMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un Direttore da un ResultSet di ritorno di una query al Database;

- Package **Esito**
  - **Esito**: classe che modella oggetti Esito e che offre metodi per leggere e modificare le relative proprietà;
  - **EsitoDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti Esito;
  - **EsitoMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un Esito da un ResultSet di ritorno di una query al Database;

- Package **Dipartimento**
  - **Dipartimento**: classe che modella oggetti Dipartimento e che offre metodi per leggere e modificare le relative proprietà. Consente di gestire i report generati per un determinato Dipartimento;
  - **DipartimentoDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti Dipartimento;
  - **DipartimentoMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un Dipartimento da un ResultSet di ritorno di una query al Database;

- Package **Report**
  - **Report**: classe che modella oggetti Report e che offre metodi per leggere e modificare le relative proprietà;
  - **ReportDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti Report;
  - **ReportMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un Report da un ResultSet di ritorno di una query al Database;
- Package **SessioneDiValidazione**
  - **SessioneDiValidazione**: classe che modella oggetti SessioneDiValidazione e che offre metodi per leggere e modificare le relative proprietà, oltre che in metodi di business necessari al funzionamento del meccanismo di generazione dei report riguardanti una sessione;

- **SessioneDiValidazioneDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti SessioneDiValidazione;
- **SessioneDiValidazioneMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un oggetto SessioneDiValidazione da un ResultSet di ritorno di una query al Database.

- Package **Formato**
  - **Formato**: classe che modella oggetti Formato e che offre metodi per leggere e modificare le relative proprietà;
  - **FormatoDAO**: classe che contiene l'implementazione dei metodi CRUD per oggetti Formato;
  - **FormatoMapper**: classe statica che fornisce metodi necessari all'estrazione dei dati di un Docente da un ResultSet di ritorno di una query al Database.

3. Class Interfaces

Di seguito, vengono elencate le interfacce delle classi previste dal Sistema, che si trovano nel package "Application Logic" e "Storage". Per una questione di leggibilità e chiarezza del documento e per evitare di aggiungere informazioni ridondanti e "rumorose", non verranno mostrate le interfacce delle classi DAO nel package "Storage" (che forniscono i metodi CRUD per l'entità di cui si fanno carico), le classi Mapper nel package "Storage" (specificate nel paragrafo 1.4 di questo documento) e alcune classi all'interno del package "Utils" nel package "ApplicationLogic". Ogni classe del Package "Storage" che ha la funzione di Bean, inoltre, è dotata di costruttori, getter e setter, che però, sempre per motivi di leggibilità e chiarezza del documento, non verranno riportati nelle interfacce delle classi.

Javadoc di Easy Pass

Per motivi di leggibilità si è scelto di creare un sito, messo in host tramite GitHub pages, contenente la Javadoc di Easy Pass. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero Sistema.

Di seguito, il link al sito in questione: https://alberto-00.github.io/Progetto-IdS/

1. Classi nel Package ApplicationLogic/Utils

Nome Classe	ConnectionSingleton
Descrizione	Questa classe rappresenta il design pattern Singleton utilizzato per effettuare e gestire le connessioni al database tramite MySQL.
Metodi	+ getConnection() : Connection + getInstance(): ConnectionSingleton +closeConnection(Connection, PreparedStatement, ResultSet): void
Invariante di classe	/

Nome Metodo	+ getConnection(): Connection
Descrizione	Questo metodo consente al Sistema di connettersi al database specificando il suo URL e di impostare i vari parametri per il setup delle connessioni.
Pre-condizioni	/
Post-condizioni	/

Nome Metodo	+ getInstance(): ConnectionSingleton
Descrizione	Questo metodo consente al Sistema di istanziare un unico oggetto ConnectionSingleton se non è stato mai creato.
Pre-condizioni	/
Post-condizioni	/

Nome Metodo	+closeConnection(Connection, PreparedStatement, ResultSet): void
Descrizione	Questo metodo consente al Sistema di chiudere le risorse passate come parametri utilizzate per la formulazione di una query al database.
Pre-condizioni	/
Post-condizioni	/

Nome Classe	Validator
Descrizione	Questa classe definisce i metodi per la validazione di richieste, conservando anche gli eventuali messaggi di errore riscontrati.
Metodi	+ hasErrors(): boolean + getErrors(): List<String> + gatherError(boolean condition, String message): boolean + required(String value): boolean + assertMatch(String value, Pattern regexp, String msg): boolean
Invariante di classe	/

Nome Metodo	+ hasErrors(): boolean
Descrizione	Questo metodo consente di verificare qualora nella richiesta siano stati riscontrati errori.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+ getError(): List<String>
Descrizione	Questo metodo restituisce la lista di errori riscontrati.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+ gatherError(boolean condition, String message): boolean
Descrizione	Se la condizione è falsa, viene aggiunto un errore alla lista.
Pre-condizione	**context** boolean :: gatherError(condition, message) **pre**: message != null
Post-condizione	/

Nome Metodo	+ required(String value): boolean
Descrizione	Questo metodo indica che la stringa in questione è obbligatoria; verifica quindi che non sia *null* o che contenga solo tabulazioni.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+ assertMatch(String value, Pattern regexp, String msg): boolean
Descrizione	Questo metodo verifica il match tra la regexp e il valore della stringa passati in input.
Pre-condizione	**context** boolean :: assertMatch(value, regexp, msg) **pre**: value != null && regexp != null && msg != null
Post-condizione	/

2. Classi nel Package Storage/PersonaleUnisa

Nome Classe	PersonaleUnisa
Descrizione	Questa è una classe astratta che modella oggetti PersonaleUnisa.
Metodi	Contiene i metodi getter e setter per le variabili d'istanza della classe: nome, cognome, e-mail, dipartimento, password. Sono tutte e quattro private di tipo String.
Invariante di classe	/

Nome Classe	DirettoreDiDipartimento
Descrizione	Questa è una classe che modella oggetti DirettoreDiDipartimento. Essa estende la classe PersonaleUnisa.
Metodi	+ eliminaReport(Report report) : void + ricercaCompletaReport(Docente docente, Date primaData, Date secondaData) : TreeMap<Report,Docente> + ricercaReportSoloDocente(Docente docente) : TreeMap<Report,Docente> + ricercaReportSoloData(Date primaData, Date secondaData): TreeMap<Report,Docente> +ricercaReport() : TreeMap<Report,Docente> + impostaFormato (Formato formato) : void
Invariante di classe	/

Nome Metodo	+ eliminaReport(Report report) : void
Descrizione	Questo metodo permette al Direttore di eliminare un report dall'insieme dei report salvati nel Dipartimento a cui appartiene.
Pre-condizione	**context**: void :: eliminaReport (report) **pre**: report != null
Post-condizione	**context**: void :: eliminaReport (report) **post**: !(dipartimento.getReports().contains(report);

Nome Metodo	+ ricercaCompletaReport(Docente docente, Date primaData, Date secondaData) : TreeMap<Report,Docente>
Descrizione	Questo metodo permette al Direttore di ricercare, nell'insieme dei report presenti nel Dipartimento a cui appartiene, tutti i report che sono stati creati dal Docente "docente" in una data compresa tra "primaData" e "decondaData".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaCompletaReport(docente, primaData, secondaData) **pre**: (docente != null && primaData!=null && secondaData!=null)
Post-condizione	/

Nome Metodo	+ ricercaReportSoloDocente(Docente docente) : TreeMap <Report,Docente>
Descrizione	Questo metodo permette al Direttore di ricercare, nell'insieme dei report presenti nel Dipartimento a cui appartiene, tutti i report che sono stati creati dal Docente "docente".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaReportSoloDocente(docente) **pre**: (docente != null)
Post-condizione	/

Nome Metodo	+ ricercaReportSoloData(Date primaData, Date secondaData) : TreeMap <Report,Docente>
Descrizione	Questo metodo permette al Direttore di ricercare, nell'insieme dei report presenti nel Dipartimento a cui appartiene, tutti i report che sono stati creati in una data compresa tra "primaData" e "decondaData".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaReportSoloData (primaData, secondaData) **pre**: (docente != null && primaData!=null && secondaData!=null)
Post-condizione	/

Nome Metodo	+ ricercaReport () : TreeMap <Report,Docente>
Descrizione	Questo metodo permette al Direttore di ricercare, nell'insieme dei report presenti nel Dipartimento a cui appartiene, tutti i report che sono stati creati dal Docente del Dipartimento a cui il Direttore è a capo.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+ impostaFormato(Formato formato) : void
Descrizione	Questo metodo consente al Direttore di impostare un formato valido per la creazione di tutti i report del Dipartimento a cui appartiene.
Pre-condizione	**context**: void :: impostaFormato(formato) **pre**: formato != null
Post-condizione	**context**: void :: impostaFormato(formato) **post**: dipartimento.getFormato() == formato

Nome Classe	Docente
Descrizione	Questa è una classe che modella oggetti Docente. Essa estende la classe PersonaleUnisa. Oltre agli attributi ereditati dalla sua superclasse, e rispettivi metodi getter e setter, questa classe possiede altre variabili d'istanza, ossia "sessioni", che serve per contenere i riferimenti agli oggetti SessioneDiValidazione e "listaReport", che contiene i riferimenti ai report che il docente ha effettuato.
Metodi	+ avviaSessione(): SessioneDiValidazione + terminaSessione(SessioneDiValidazione sessione): void
Invariante di classe	/

Nome Metodo	+ avviaSessione(): SessioneDiValidazione
Descrizione	Questo metodo consente al Docente di iniziare una sessione di validazione per il controllo dei Green Pass.
Pre-condizione	/
Post-condizione	**context**: SessioneDiValidazione :: avviaSessione() **post**: sessioni.contains(SessioneDiValidazione:: avviaSessione())

Nome Metodo	+ terminaSessione(SessioneDiValidazione sessione): void
Descrizione	Questo metodo consente al Docente di terminare una sessione di validazione, precedentemente avviata.
Pre-condizione	**context**: void :: terminaSessione(sessione) **pre**: sessione != null
Post-condizione	/

3. Classi nel Package Storage/Dipartimento

Nome Classe	Dipartimento
Descrizione	Questa classe modella oggetti Dipartimento. I suoi attributi sono nome, codice, un oggetto Formato e una lista di report.
Metodi	+ impostaFormato(Formato formato) : void + eliminaReport(Report report) : void + ricercaCompletaReport(Docente docente, Date primaData, Date secondaData) : TreeMap<Report,Docente> + ricercaReportSoloDocente(Docente docente) : TreeMap <Report,Docente> + ricercaReportSoloData(Date primaData, Date secondaData) : TreeMap<Report,Docente> +ricercaReport() : TreeMap<Report,Docente>
Invariante di classe	/

Nome Metodo	+ impostaFormato(Formato formato) : void
Descrizione	Questo metodo consente di aggiornare il formato dei report generati.
Pre-condizione	**context**: void :: impostaFormato(formato) **pre**: formato != null
Post-condizione	/

Nome Metodo	+ eliminaReport(Report report) : void
Descrizione	Questo di eliminare un report dall'insieme dei report salvati in questo Dipartimento.
Pre-condizione	**context**: void :: eliminaReport (report) **pre**: report != null
Post-condizione	**context**: void :: eliminaReport (report) **post**: !(this.getReports().contains(report);

Nome Metodo	+ ricercaCompletaReport(Docente docente, Date primaData, Date secondaData) : TreeMap<Report,Docente>
Descrizione	Questo metodo permette di ricercare, nell'insieme dei report presenti in questo Dipartimento, tutti i report che sono stati creati dal Docente "docente" in una data compresa tra "primaData" e "decondaData".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaCompletaReport(docente, primaData, secondaData) **pre**: (docente != null && primaData!=null && secondaData!=null)
Post-condizione	/

Nome Metodo	+ ricercaReportSoloDocente(Docente docente) : TreeMap <Report,Docente>
Descrizione	Questo metodo permette di ricercare, nell'insieme dei report presenti in questo Dipartimento, tutti i report che sono stati creati dal Docente "docente".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaReportSoloDocente(docente) **pre**: (docente != null)
Post-condizione	/

Nome Metodo	+ ricercaReportSoloData(Date primaData, Date secondaData) : TreeMap <Report,Docente>
Descrizione	Questo metodo permette di ricercare, nell'insieme dei report presenti in questo Dipartimento, tutti i report che sono stati creati in una data compresa tra "primaData" e "decondaData".
Pre-condizione	**context**: TreeMap<Report,Docente> :: ricercaReportSoloData (primaData, secondaData) **pre**: (docente != null && primaData!=null && secondaData!=null)
Post-condizione	/

Nome Metodo	+ ricercaReport () : TreeMap <Report,Docente>
Descrizione	Questo metodo permette di ricercare tutti i report generati in questo Dipartimento, memorizzando per ognuno anche il Docente che lo ha creato.
Pre-condizione	/
Post-condizione	/

4. Classi nel Package Storage/Report

Nome Classe	Report
Descrizione	Questa classe modella oggetti Report. I suoi attributi sono id, data, orario, pathFile, un oggetto Dipartimento e un oggetto Docente.
Metodi	+ creaFile(ArrayList<Esito> esiti): void - zebraRow(Document documento, PdfPTable tabella):void + toJson(): JSONObject
Invariante di classe	/

Nome Metodo	+ creaFile(ArrayList<Esito> esiti): void
Descrizione	Questo metodo consente di creare e memorizzare sul server un file in formato pdf che rappresenta un Report contenente la lista di esiti passata come parametro.
Pre-condizioni	**context**: void :: creaFile(esiti) **pre**: (esiti != null)
Post-condizioni	/

Nome Metodo	- zebraRow(Document documento, PdfPTable tabella):void
Descrizione	Questo metodo consente di creare una tabella che verrà inserita all'interno del file pdf del Report, formattandola in modo che le righe siano di due colori alternati.
Pre-condizioni	/
Post-condizioni	/

Nome Metodo	- toJson(): JSONObject
Descrizione	Questo metodo consente di trascrivere il Report in formato JSON.
Pre-condizioni	/
Post-condizioni	/

5. Package SessioneDiValidazione

Nome Classe	SessioneDiValidazione
Descrizione	Questa classe modella oggetti SessioneDiValidazione. I suoi attributi sono QRCode, isInCorso, url, una ArrayList di Esiti e un oggetto Docente.
Metodi	- creaQRCode(String url): BufferedImage + validaGreenPass(Esito esito) : Esito
Invariante di classe	/

Nome Metodo	- creaQRCode(String url): BufferedImage
Descrizione	Questo metodo genera un'immagine che rappresenta un codice QR che rimanda all'indirizzo specificato in "url"
Pre-condizioni	/
Post-condizioni	/

Nome Metodo	+ validaGreenPass(Esito esito) : Esito
Descrizione	Questo metodo invia la stringa che rappresenta il GP, contenuta dall'oggetto "esito", al ValidationServer che risponderà con l'esito della validazione
Pre-condizioni	**context**: Esito:: validaGreenPass (esito) **pre**: esito.getStringaGP() != null
Post-condizioni	/

6. Classi nel Package Esito

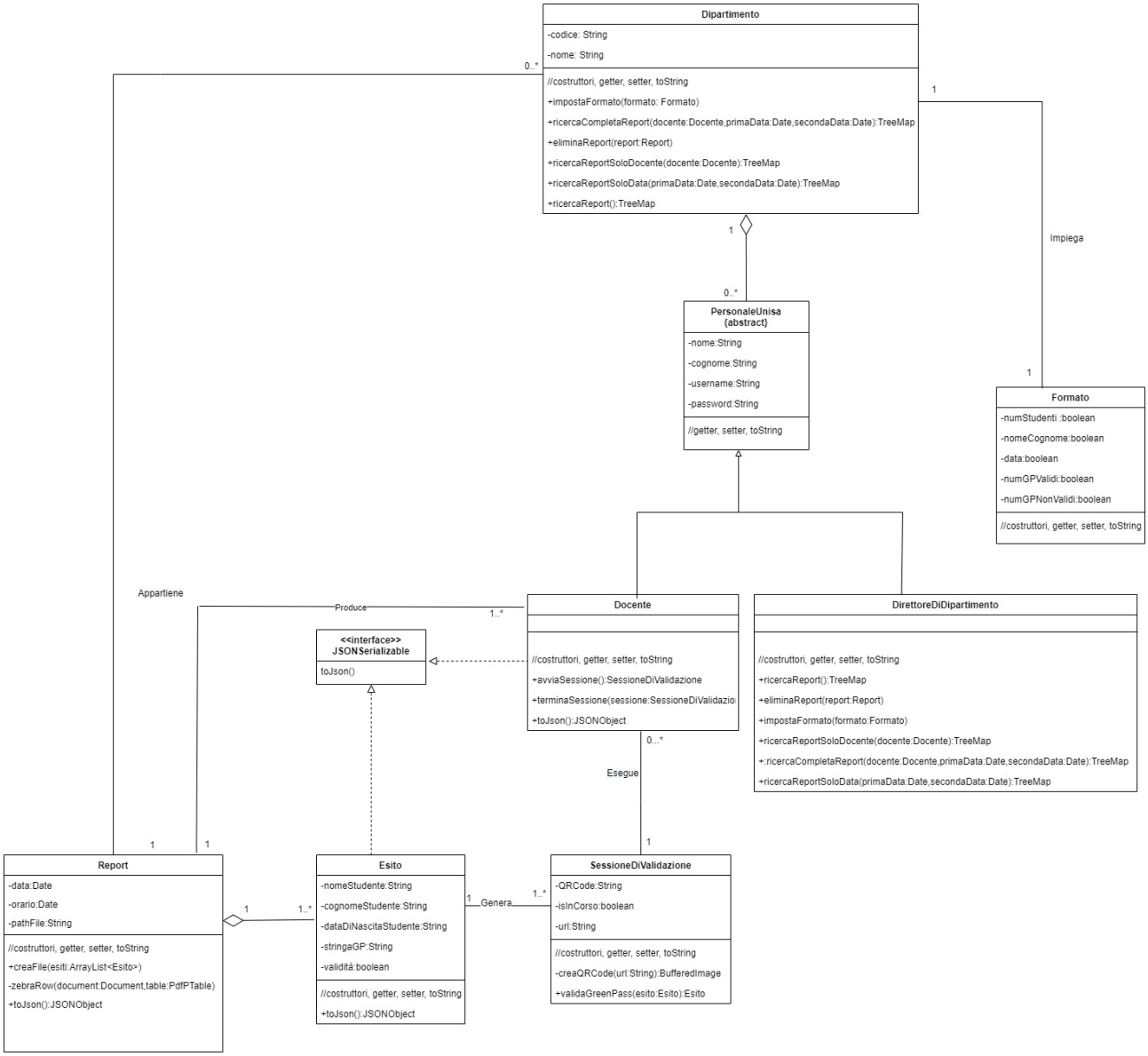
Nome Classe	Esito
Descrizione	Questa classe modella oggetti Esito. I suoi attributi sono id, nomeStudente, cognomeStudente, dataDiNascitaStudente, stringaGP, validita, un oggetto Report e un oggetto SessioneDiValidazione.
Metodi	+ toJson(): JSONObject
Invariante di classe	/

Nome Metodo	- toJson(): JSONObject
Descrizione	Questo metodo consente di trascrivere l'Esito in formato JSON.
Pre-condizioni	/
Post-condizioni	/

7. Classi nel Package Formato

Nome Classe	Formato
Descrizione	Questa classe modella oggetti Formato. I suoi attributi sono id, numStudenti, nomeCognome, data, numGPValidi, numGPNonValidi.
Metodi	/
Invariante di classe	/

4. Class Diagram



Il class diagram qui riportato presenta delle differenze e delle migliorie rispetto a quello mostrato nel documento RAD. Le fasi dello sviluppo del "Modello ad oggetti" hanno portato alla realizzazione di un Class Diagram evoluto. Inoltre, è stata utilizzata l'ereditarietà di specifica e di implementazione per astrarre i concetti che accomunano Docente e DirettoreDiDipartimento e sono stati aggiunti metodi e attributi alle classi. Si è scelto inoltre di utilizzare la tecnica di mapping orizzontale per tradurre le classi

della gerarchia "PersonaleUnisa", "Docente" e "DirettoreDiDipartimento", e ciò spiega la mancata presenza di tale gerarchia nel modello concettuale del database mostrato nel documento SDD.

5. Design Pattern

In questa sezione del presente documento si andranno a descrivere e dettagliare i design pattern utilizzati nello sviluppo della Web Application Easy Pass. Per ogni pattern si darà:

- ✓ una breve introduzione teorica;
- ✓ il problema che doveva essere risolto all'interno di Easy Pass;
- ✓ una breve spiegazione di come si è risolto il problema in Easy Pass.

Singleton

Il Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell'istanziazione degli oggetti, che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza e di fornire un punto di accesso globale a tale istanza.

Easy Pass prevede un sistema per connettersi al database MySQL tramite la creazione di un unico oggetto Connection per evitare di saturare la memoria.

Per gestire ciò, viene creato un oggetto Singleton che conserva un'istanza della classe "ConPool.java" con lo scopo di effettuare le connessioni al database MySQL.

Adapter

L'Adapter è un design pattern strutturale, ovvero quei design pattern che facilitano la progettazione attraverso la semplificazione delle relazioni tra entità. L'Adapter, in particolare, permette ad oggetti con differenti interfacce di collaborare. Si implementa attraverso una classe "adapter" che si occupa di convertire i dati in oggetti comprensibili dal Sistema.

Easy Pass si pone l'obiettivo di semplificare e velocizzare la validazione dei Green Pass da parte dei Docenti, per questo il Sistema offre la funzionalità di validazione automatica dei Green Pass che vengono inviati. Per effettuare la validazione viene utilizzata una libreria fornita dal Ministero della Salute chiamata "dcc-utils" che è scritta in node.JS, mentre il back-end del nostro Sistema è scritto in Java. Poiché il linguaggio nodeJS permette facilmente di creare server HTTP, si è deciso di inglobare l'intera procedura

di validazione in un server chiamato "ValidationServer" e di utilizzare i pacchetti HTTP come principale metodo di comunicazione tra i due sistemi.

Il design pattern Adapter si occupa della conversione dei pacchetti HTTP in oggetti interpretabili dai rispettivi linguaggi. Quando la comunicazione avviene da Easy Pass al ValidationServer, viene scritta la stringa rappresentante il Green Pass come parametro dell'URL che poi viene letta e convertita in DCC (Digital Covid Certificate). Quando la validazione è terminata, l'esito della stessa viene inserito nell'header della response HTTP che poi viene letta e convertita in oggetto Esito grazie all'adapter.

DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con necessità di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali.

Essendo Easy Pass una Web Application con l'obiettivo di generare numerosi report tramite la validazione dei Green Pass, sono necessarie molte interazioni con il database, quindi, il Sistema ha bisogno di poter interagire con questo in modo rapido e sicuro effettuando numerose query.

Per gestire ciò, vengono create classi che implementano metodi contenenti ciascuno una connessione al database e una query con successive istruzioni.

6. Glossario

In questa sezione descriveremo i termini tecnici che sono stati utilizzati all'interno del Documento stesso.

- **GitHub**: servizio di hosting per progetti software;
- **Web Application**: applicazione accessibile via Web per mezzo di una rete Internet.