****

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **O**bject **D**esign  **D**ocument  Easy Pass   |  |  | | --- | --- | | Riferimento |  | | Versione | 1.0 | | Data | 9/12/2021 | | Destinatario | Prof. C. Gravino | | Presentato da | Montefusco Alberto  Mulino Martina  Rinaldi Viviana  Spina Gennaro | | Approvato da |  | |

Sommario

[Sommario 2](#_Toc530825397)

[RevisionHistory 3](#_Toc530825396)

[Responsabilità 4](#_Toc530825396)

1. [Introduzione 5](#_Toc530825398)

1.1 Object Design Goals 5

1.2 Object Design Trade-off 5

1.3 Componenti off-the-shelf 5

1.4 Linee guida per la documentazione di Interfacce ........................................................................... 7

1.5 Definizioni, acronimi e abbreviazioni ............................................................................................... 8

1.6 Riferimenti ............................................................................................................................................ 9

1. Packages ........................................................................................................................................................... 9

2.1 Interface ................................................................................................................................................ 9

2.2 Application Logic ................................................................................................................................. 9

2.3 Storage ................................................................................................................................................... 9

1. Class Interfaces ................................................................................................................................................ 9
2. Class Diagram................................................................................................................................................ 18
3. Design Pattern ............................................................................................................................................... 18
4. Glossario ........................................................................................................................................................ 20

RevisionHistory

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

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 | Martina Mulino |
| Design Pattern | Alberto Montefusco, Viviana Rinaldi, Martina Mulino, Gennaro Spina |
| Class Interfaces: Esito, EsitoDAO, Dipartimento, DipartimentoDAO | Alberto Montefusco |
| Class Interfaces: DirettoreDiDipartimento, DirettoreDiDipartimentoDAO, Docente, DocenteDAO | Martina Mulino |
| Class Interfaces: Report, ReportDAO, SessioneDiValidazione, SessioneDiValidazioneDAO | Gennaro Spina |
| Class Interfaces: Formato, FormatoDAO, SessionController, AccessController, ReportController | Viviana Rinaldi |

1. Introduzione

L’Object Design Document illustra i diversi dettagli legati alla fase implementativa del Sistema Easy Pass; in particolare, esso descrive i 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. 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. 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. 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.
  1. Linee guida per la documentazione di Interfacce

Tali linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce. Per la loro realizzazione si è fatto riferimento alla convenzione Java nota come **Sun Java Coding Convention** [Sun, 2009]. Di seguito sono riportati due link alle convenzioni usate per definire le linee guida:

1. Java Sun: https://checkstyle.sourceforge.io/sun\_style.html
2. 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 e DAO).
* Ogni classe che funge da Bean, all’interno del package Storage, deve contenere 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 package Interface contiene dei sub-package in cui sono organizzati tutti i file che si occupano dell’interfaccia utente (JSP, pagine HTML).
* 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. 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.

1. **Acronimi**:

* **SDD**: System Design Document
* **RAD**: Requirements Analysis Document
  1. 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.

1. 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.

* 1. Interface

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

* Package **StudenteGUI**
  + **SendGP**: visualizza il form per l’inserimento del Green Pass.
* Package **DocenteGUI**
  + **AvviaSessione**: 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 in corso;
  + **Registrazione**: mostra il form di registrazione per un docente.
* 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 le check-box necessarie alla scelta.
* Package **AutenticazioneGUI**
  + **Login**: fornisce il form di login per Direttore e 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**
  + **Footer**
  1. Application Logic

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

* 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**
  + **RequestValidator**
  + **ConnectionSingleton**
  1. 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 estende “PersonaleUnisa” e che modella un Docente;
  + **DocenteDAO**: classe che contiene l’implementazione dei metodi CRUD per oggetti Docente;
  + **DocenteMapper**: classe statica che fornisce 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;

1. Class Interfaces

Di seguito, vengono elencate tutte le classi previste dal Sistema, in particolare dei package Application Logic e Storage.

**Javadoc di Easy Pass**

Per motivi di leggibilità si è scelto di creare un sito, hostato 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://stefanolambiase.github.io/biblionet/>

1. **Package Utils**

|  |  |
| --- | --- |
| **Nome Classe** | **ConnectionSingleton** |
| **Descrizione** | Questa classe rappresenta il design pattern Singleton utilizzato per effettuare e gestire le connessioni al database MySQL tramite. |
| **Metodi** | + getConnection() : Connection  + getIstance(): ConnectionSingleton |
| **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** | **+ getIstance(): ConnectionSingleton** |
| **Descrizione** | Questo metodo consente al Sistema di istanziare un unico oggetto ConnectionSingleton se non è stato mai creato. |
| **Pre-condizioni** | / |
| **Post-condizioni** | / |

|  |  |
| --- | --- |
| **Nome Classe** | **RequestValidator** |
| **Descrizione** | Questa classe definisce metodi per la validazione di richieste, conservando anche gli eventuali errori riscontrati. |
| **Metodi** | + hasErrors(): boolean  + getErrors(): List<String>  + gatherErrors(boolean condition, Sting 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** | **+ getErrors(): List<String>** |
| **Descrizione** | Questo metodo restituisce la lista di errori riscontrati. |
| **Pre-condizione** | / |
| **Post-condizione** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+ addError(boolean condition, String message): boolean** |
| **Descrizione** | Se la condizione è falsa, viene aggiunto un errore alla lista. |
| **Pre-condizione** | **context** boolean :: addError(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** | / |

1. **Package 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) : Report  + downloadReport(Report report) : boolean  + ricercaReport(String docente, Date primaData, Date secondaData) : List<Report>  + impostaFormato (Formato formato) : Formato; |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+ eliminaReport(Report report) : Report** |
| **Descrizione** | Questo metodo permette al Direttore di eliminare un report dall’insieme dei report salvati nel Dipartimento a cui appartiene. |
| **Pre-condizione** | **context**: Report :: eliminaReport (report)  **pre**: report != null |
| **Post-condizione** | **context**: Report :: eliminaReport (report)  **post**: !(dipartimento.getReports().contains(report); |
| **Nome Metodo** | **+ downloadReport(Report report) : boolean** |
| **Descrizione** | Questo metodo permette al Direttore di effettuare il download di un report sulla propria macchina. |
| **Pre-condizione** | **context**: boolean :: downloadReport (report)  **pre**: report != null |
| **Post-condizione** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+ ricercaReport(String docente, Date primaData, Date secondaData) : List<Report>** |
| **Descrizione** | Questo metodo permette al Direttore di ricercare alcuni report nell’insieme dei report presenti nel Dipartimento a cui appartiene, specificando dei parametri di ricerca. |
| **Pre-condizione** | **context**: List<Report> :: ricercaReport(docente, primaData, secondaData)  **pre**: (docente != null || primaData!=null) |
| **Post-condizione** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+ impostaFormato(Formato formato)** |
| **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**: Formato :: impostaFormato(formato)  **pre**: formato != null |
| **Post-condizione** | **context**: Formato :: 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 un’altra variabile d’istanza, chiamata “sessioni”, che serve per contenere i riferimenti agli oggetti SessioneDiValidazione. |
| **Metodi** | + avviaSessione(): SessioneDiValidazione  + terminaSessione(SessioneDiValidazione sessione): boolean  + downloadReport(Report report): boolean |
| **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): boolean** |
| **Descrizione** | Questo metodo consente al Docente di terminare una sessione di validazione, precedentemente avviata. |
| **Pre-condizione** | **context**: boolean :: terminaSessione(sessione)  **pre**: sessione != null |
| **Post-condizione** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+ downloadReport(Report report): boolean** |
| **Descrizione** | Questo metodo consente al Docente di effettuare il download del report relativo alla sessione appena creata. |
| **Pre-condizione** | **context**: boolean :: downloadReport(report)  **pre**: report != null |
| **Post-condizione** | / |

1. **Package Dipartimento**

|  |  |
| --- | --- |
| **Nome Classe** | **Dipartimento** |
| **Descrizione** | Questa classe modella oggetti Dipartimento. |
| **Metodi** | + impostaFormato(Formato formato) : Formato |
| **Invariante di classe** | / |

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

1. Class Diagram
2. Design Pattern

In questa sezione del presente documento si andranno a descrivere e dettagliare i design patterns 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;
* un grafico della struttura delle classi che implementano il pattern.

**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.

Grafico struttura

**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, è necessario un database molto vasto, 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.

Grafico struttura

1. Glossario