Gestione Web Ritardi Allievi SAMT

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 4

2 Analisi 5

2.1 Analisi del dominio 5

2.2 Analisi e specifica dei requisiti 5

2.3 Use case 9

2.4 Pianificazione 10

2.4.1 Analisi 11

2.4.2 Progettazione 11

2.4.3 Implementazione 12

2.4.4 Testing 12

2.4.5 Consegna 12

2.5 Analisi dei mezzi 13

2.5.1 Software 13

2.5.2 Hardware 13

3 Progettazione 14

3.1 Design dell’architettura del sistema 14

3.2 Design dei dati e database 14

3.2.1 Schema ER 14

3.2.2 Descrizioni delle tabelle 15

3.2.3 Schema logico 18

3.3 Design delle interfacce 18

3.3.1 Pagina di accesso 18

3.3.2 Pagina cambio password 18

3.3.3 Pagina di recupero password 20

3.3.4 Pagina di aggiunta ritardi 20

3.3.5 Pagina di aggiunta recuperi 21

3.3.6 Pagina di gestione utenti 21

3.3.7 Pagina impostazioni 22

4 Implementazione 22

4.1 Gestione versioni 22

4.2 Gestore di pacchetti 22

4.3 Database 23

4.3.1 Account di accesso 23

4.3.2 Implementazione banca dati 23

4.3.3 Interrogazione database 24

4.4 Applicativo web 24

4.4.1 Struttura 24

4.4.2 Model View Controller (MVC) 25

4.4.3 Representational State Transfer (REST) 26

4.4.4 Routing delle richieste 26

4.4.5 Configurazione 27

4.4.6 Autenticazione 27

4.4.6.1 Gestione delle sessioni 28

4.4.6.2 Gestione dei permessi 29

4.4.6.3 Gestione percorsi e accessi 30

4.4.7 Invio di e-mail 31

4.4.8 Validazione dei dati 31

4.4.9 Gestione dati ed interrogazione database 31

4.4.9.1 Tabella user 32

4.4.9.2 Tabella token 34

4.4.9.3 Tabella student 34

4.4.9.4 Tabella section 35

4.4.9.5 Tabella delay 36

4.4.9.6 Tabella year 36

4.4.9.7 Tabella setting 36

4.4.10 Creazione PDF 38

4.4.11 Sicurezza 38

4.4.11.1 Interrogazione database 38

4.4.11.2 Salvataggio dei dati 39

4.4.11.3 Salvataggio credenziali 39

4.4.11.4 Recupero password 40

4.4.12 Interfacce grafiche 40

4.4.12.1 Pagina di accesso 40

4.4.12.2 Pagina di cambio password 40

4.4.12.3 Pagina di recupero password 40

4.4.12.4 Pagina di gestione ritardi 40

4.4.12.5 Pagina di gestione recuperi 40

4.4.12.6 Pagina di gestione utenti 40

4.4.12.7 Pagina impostazioni 40

5 Test 41

5.1 Protocollo di test 41

5.2 Risultati test 41

5.3 Mancanze/limitazioni conosciute 41

6 Consuntivo 41

7 Conclusioni 41

7.1 Sviluppi futuri 41

7.2 Considerazioni personali 41

8 Sitografia 41

9 Allegati 41

# Introduzione

## Informazioni sul progetto

**Titolo:** Gestione Web Ritardi Allievi SAMT  
**Allievo coinvolto nel progetto:** Filippo Finke, [filippo.finke@samtrevano.ch](mailto:filippo.finke@samtrevano.ch)

**Classe:** I4AC Scuola Arti e Mestieri Trevano, Informatica

**Formatore:** Fabrizio Valsangiacomo, [fabrizio.valsangiacomo@edu.ti.ch](mailto:fabrizio.valsangiacomo@edu.ti.ch)

**Perito:** Gianluca Costante, [gianluca.costante@gmail.com](mailto:gianluca.costante@gmail.com)

**Data inizio:** 11.05.2020

**Data fine:** 29.05.2020

**Durata:** 80 ore

DA\_REVISIONARE

## Abstract

*The aim of the project "SAMT Student Web Delays Management" is to simplify, speed up and automate the process of the assignment of the students delays at the SAMT. At the moment the delays are marked mainly on paper or with other methods that do not guarantee the integrity of the data themselves and do not allow to perform quick searches, controls and other. The project in question therefore deals with digitizing what is done manually and creating a website that allows you to manage everything in an automated way. This will speed up the work to be done by the teachers. Thanks to this system it will also be possible to keep a record of the accumulated and recovered delays of all the students who have been registered. You will also be able to create a PDF file with the history of a student.*

DA\_REVISIONARE

## Scopo

Lo scopo del progetto “Gestione Web Ritardi Allievi SAMT” è quello di semplificare, velocizzare ed automatizzare il processo di assegnazione dei ritardi all’interno della Scuola d’Arti e Mestieri di Trevano. Al momento i ritardi vengono segnati principalmente su carta o con altre metodologie che però non garantiscono l’integrità dei dati stessi e non permettono di eseguire ricerche velocemente, controlli e altro. Il progetto in questione si occupa dunque di digitalizzare ciò che viene fatto manualmente e di creare un sito web che permetta di gestire il tutto in modo automatizzato. Questo permetterà di velocizzare il lavoro che dovrà essere svolto dai docenti. Grazie a questo sistema si potrà inoltre tenere uno storico dei ritardi accumulati e recuperati di tutti gli studenti che sono stati registrati. Sarà inoltre possibile creare un file PDF con lo storico di uno studente.

DA\_REVISIONARE

# Analisi

## Analisi del dominio

È stato richiesto lo sviluppo di un gestionale web per la gestione dei ritardi della Scuola d’Arti e Mestieri di Trevano. Il prodotto dovrà essere un applicativo web accessibile attraverso la rete utilizzando browser moderni (Esempio: Chrome). Gli utenti che accederanno a questo applicativo saranno principalmente docenti di classe che andranno ad inserire e gestire i ritardi dei propri alunni. Saranno disponibili ulteriori permessi che permetteranno di distinguere gli utenti in utenti amministratori e utenti normali. Gli utenti amministratori potranno gestire gli utenti presenti all’interno dell’applicativo. Gli utenti normali avranno dei permessi limitati (inserimento dei ritardi, visione dei dati o creazione PDF) che permetteranno a questo utente di eseguire solamente determinate azioni limitate. Sarà presente una pagina di amministrazione che permetterà agli amministratori di modificare alcune impostazioni dell’applicativo come ad esempio la soglia limite dei ritardi prima del recupero e altre opzioni. I docenti di classe potranno inserire studenti ed i relativi ritardi all’interno del sistema, sarà inoltre possibile inserire quando uno studente ha recuperato un determinato ritardo in modo da poterne tenere uno storico. Vi sarà inoltre la possibilità di inserire dei ritardi giustificati, ovvero dei ritardi che vengono mostrati nel conteggio totale ma non dovranno essere recuperati dallo studente. Una volta raggiunta la soglia massima di ritardi un sistema automatico invierà una e-mail di notifica allo studente informandolo che verrà contattato per eseguire il recupero del ritardo. Ogni semestre verrà eseguito un reset automatico del numero di ritardi degli studenti ma non dei ritardi da recuperare. L’applicativo dovrà inoltre essere caricato su un hosting.

DA\_REVISIONARE

## Analisi e specifica dei requisiti

È richiesto da parte del committente lo sviluppo di un applicativo web che sia accessibile tramite la rete e che sia web. Il prodotto dovrà possedere un sistema di autenticazione con un sistema di permessi integrato. Il sistema dovrà dunque distinguere i vari utenti in base ai loro permessi. È dunque richiesta una pagina di login attraverso la quale gli utenti andranno ad accedere all’applicativo. Gli amministratori del software potranno aggiungere ed eliminare gli utenti, alla creazione di un utente le credenziali per l’accesso verranno inviate in modo automatico attraverso un messaggio di posta elettronica e al primo accesso verrà richiesto all’utente di cambiare la propria password. In caso di perdita della propria password essa potrà essere recuperata attraverso il proprio indirizzo e-mail. Gli amministratori potranno aggiungere utenti amministratori oppure utenti limitati i quali potranno avere i seguenti permessi:

* Inserimento dei ritardi
* Visione dei dati
* Creazione dei PDF

Dovrà essere presente una pagina di amministrazione attraverso la quale sarà possibile configurare le date di inizio e di fine di ogni semestre, i nomi delle sezioni scolastiche, gli anni scolastici e la soglia di ritardi massimi dopo i quali essi andranno recuperati. I docenti avranno a disposizione una pagina attraverso la quale potranno registrare gli allievi che hanno cominciato ad accumulare dei ritardi e di conseguenza sarà possibile anche aggiungere ritardi ad allievi già registrati. Sarà disponibile un'altra pagina che permetterà ai docenti di inserire i ritardi recuperati da parte degli studenti. Di ogni studente verrà inoltre tenuto lo storico dei ritardi. Il prodotto dovrà essere caricato su un hosting.

DA\_REVISIONARE

|  |  |
| --- | --- |
| **ID: REQ-000** | |
| **Nome** | Piattaforma dell’applicativo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di un applicativo web. |
| **Sotto requisiti** | |
| **001** | Il sito deve funzionare su browser moderni (Esempio: Chrome). |

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Pagina di accesso |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di inserire le proprie credenziali per accedere all’applicativo web. |
| **Sotto requisiti** | |
| **001** | Si necessita di una maschera di login. |
| **002** | Si necessita di un modale di cambio password per nuovi utenti. |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Pagina di recupero password |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di recuperare la propria password. |
| **Sotto requisiti** | |
| **001** | Dovrà essere inviata una e-mail di recupero password se l’utente che la richiede è esistente. |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Pagina di gestione utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di gestire gli utenti che avranno accesso all’applicativo web. |
| **Sotto requisiti** | |
| **001** | Dovrà essere possibile creare utenti. |
| **002** | Le credenziali verranno inviate per posta elettronica. |
| **003** | Dovrà essere possibile eliminare utenti. |
| **004** | Dovrà essere possibile modificare i permessi di ogni singolo utente. |
| **005** | Dovrà essere sempre presente almeno un utente amministratore. |
| **006** | Non dovrà essere possibile eliminare il proprio utente. |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Pagina di amministrazione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di gestire le impostazioni dell’applicativo web. |
| **Sotto requisiti** | |
| **001** | Dovrà essere possibile inserire le date di inizio e fine di ogni semestre. |
| **002** | Dovrà essere possibile inserire i nomi delle sezioni. |
| **003** | Dovrà essere possibile inserire gli anni scolastici. |
| **004** | Dovrà essere possibile modificare la soglia di ritardi massimi. (Di base tre compreso) |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Pagina di aggiunta ritardi |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di aggiungere ritardi agli studenti. |
| **Sotto requisiti** | |
| **001** | Dovrà essere possibile inserire nuovi studenti. |
| **002** | Dovrà essere possibile aggiungere un ritardo ad uno studente. |
| **003** | Al raggiungimento della soglia massima dovrà essere inviata una e-mail di notifica. |
| **004** | Dovrà essere tenuto un istoriato dei ritardi. |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Pagina di gestione recuperi |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina che permetta di aggiungere i ritardi recuperati da parte degli studenti. |
| **Sotto requisiti** | |
| **001** | Dovrà essere possibile inserire il ritardo recuperato da parte dello studente. |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Messa in produzione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’applicativo dovrà essere messo in produzione su un hosting. |

DA\_REVISIONARE

## Use case

Gli agenti di questo schema si dividono in docente ed amministratore. Entrambi gli agenti hanno la possibilità di accedere alla pagina di accesso attraverso la quale sarà possibile accedere all’applicativo. Attraverso la pagina di login sarà possibile recarsi in un'altra pagina dedicata al recupero password. Entrambi gli agenti potranno eseguire azioni di gestione di recuperi e ritardi ovvero potranno creare degli studenti ed assegnarci dei ritardi. Ulteriormente potranno anche gestire i recuperi dei ritardi eseguiti dagli studenti. Successivamente solo gli amministratori potranno accedere alle pagine di gestione degli utenti dove sarà possibile aggiungere, modificare ed eliminare gli utenti presenti nell’applicativo. L’amministratore potrà accedere ad una ulteriore pagina di amministrazione che permetterà di modificare le impostazioni dell’applicativo come: i semestri, gli anni scolastici, le sezioni e la soglia massima dei ritardi. In questo caso l’utente docente ha tutti i permessi per accedere alle pagine di gestione dei ritardi, prima di tutte queste pagine viene eseguito un controllo dei permessi dell’utente per decidere se esso può accedere oppure no ad una determinata pagina. I permessi controllati saranno:

* Inserimento dei ritardi
* Visione dei dati
* Creazione dei PDF

DA\_REVISIONARE

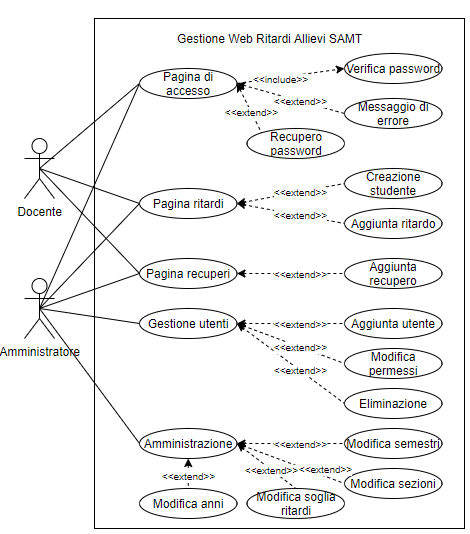


Figura Schema caso d'uso.

## Pianificazione

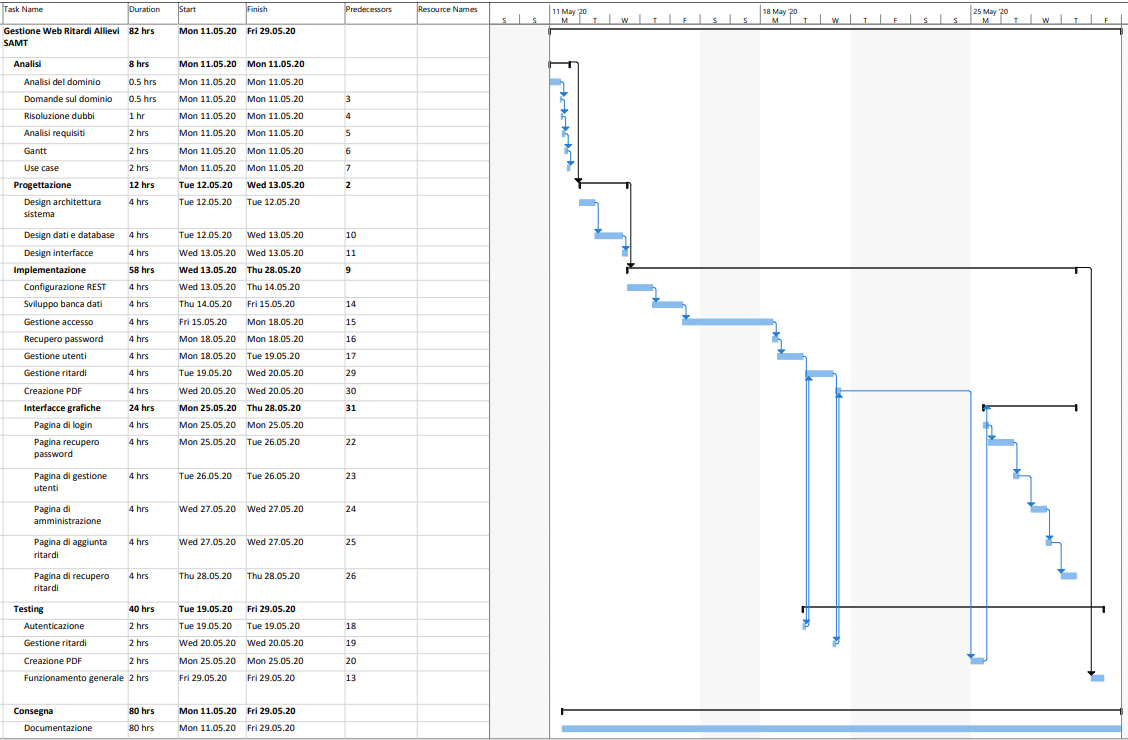


Figura Diagramma di Gantt preventivo.

### Analisi

Ho suddiviso la fase di analisi in sei attività. Durante questa fase mi sono occupato di capire di cosa tratta il progetto e quali sono le richieste da parte del committente.

DA\_REVISIONARE

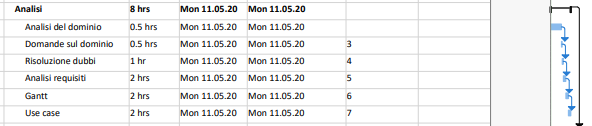


Figura Diagramma di Gantt, Analisi.

### Progettazione

La progettazione è la seconda fase del progetto “Gestione Web Ritardi Allievi SAMT” ed è composta da tre attività. Questa fase contiene le attività di design della struttura del progetto. Considero questa fase molto importante in quanto le fasi successive saranno basate su di essa.

DA\_REVISIONARE

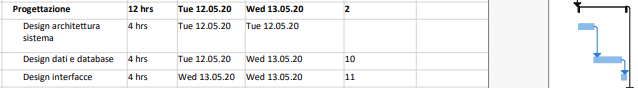


Figura Diagramma di Gantt, Progettazione.

### Implementazione

La fase di implementazione è la più lunga all’interno del progetto. Questa fase consiste nella vera scrittura del codice che andrà a comporre il prodotto finale. Questa fase è strettamente collegata all’analisi e alla progettazione in quanto si basa su di esse per lo sviluppo.

DA\_REVISIONARE

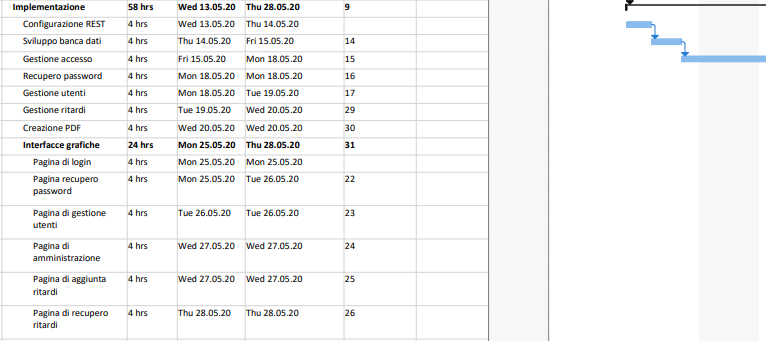


Figura Diagramma di Gantt, Implementazione.

### Testing

Un’altra fase molto importante è il testing. All’interno di questa fase sono presenti i test che verranno eseguiti alle varie sezioni del progetto. Le attività di testing sono state generalizzate in modo tale da rendere il gantt leggibile, durante il corso del progetto verranno eseguiti test più approfonditi.

DA\_REVISIONARE



Figura Diagramma di Gantt, Testing.

### Consegna

L’ultima fase del diagramma è la consegna del prodotto, all’interno di essa vi è una sola attività che riguarda la documentazione. Questa attività ha la durata dell’interno progetto perché verrà aggiornata nel corso dello sviluppo di esso.

DA\_REVISIONARE



Figura Diagramma di Gantt, Consegna.

## Analisi dei mezzi

### Software

I software utilizzati per la realizzazione del progetto sono:

- Google Chrome 76.0

- Microsoft Word 265 18.2004

- Microsoft Project 2019

- Microsoft VS Code 1.37.1

- MySQL 8.0.13

- PHP 7.3.5

- Draw.io (<https://draw.io>)

- PaperCut 5.7.0

- Postman 7.24.0

Librerie utilizzate:

- jQuery 3.4.1 (https://jquery.com/)

- Bootstrap 4.3.1 (https://getbootstrap.com/)

- php-rest (<https://github.com/filippofinke/php-rest>)

- FPDF (<http://www.fpdf.org/>)

Template utilizzati:

- SB Admin 2 (https://github.com/BlackrockDigital/startbootstrap-sb-admin-2)

Gestore librerie utilizzato:

- Composer 1.7.3 (<https://getcomposer.org/download/>)

DA\_REVISIONARE

### Hardware

Il progetto è stato sviluppato su un computer fisso.

Le specifiche hardware sono:

- 32 GB di RAM

- Intel Core I7-7700K 4 core

Il progetto potrà essere messo in produzione su una qualsiasi macchina con più di:

- 512MB di RAM

- 2GB di disco

DA\_REVISIONARE

# Progettazione

## Design dell’architettura del sistema

Questo è lo schema dell’architettura del sito web molto semplificato, l’applicativo web interagisce con il database in due modi. Il primo modo è diretto tramite l’utilizzo dei metodi messi a disposizione da PHP per l’interrogazione della banca dati MySQL. Mentre il secondo modo per interrogare la banca dati è attraverso delle richieste HTTP eseguite in modo asincrono da JavaScript. Per implementare questo sistema vengono dunque utilizzate due stili architetturali: REST e MVC. Vengono ricavati i dati in modo diretto in azioni che non vanno ad influenzare una buona esperienza utente, mentre dove vengono eseguite delle azioni da parte dell’utente (ad esempio l’aggiunta di un utente) vengono eseguite delle chiamate alle API REST in modo da offrire una migliore esperienza all’utente.

DA\_REVISIONARE

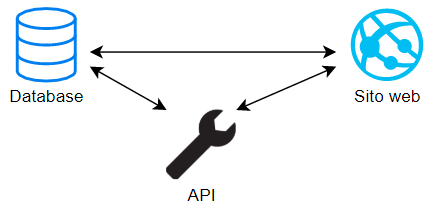


Figura Architettura del sistema semplificata.

## Design dei dati e database

Tutti i dati riguardanti l’applicativo verranno salvati all’interno di un database MySQL.

DA\_REVISIONARE

### Schema ER

Questo è lo schema ER utilizzato dall’applicativo web. È composto da sette tabelle. La tabella “USER” viene utilizzata per salvare tutte le informazioni relative agli utenti che potranno utilizzare l’applicativo web. La tabella “TOKEN” viene utilizzata per immagazzinare i codici di recupero password che vengono generati quando un utente dimentica la propria password. La tabella “STUDENT” contiene tutti gli studenti che hanno fatto almeno un ritardo e che dunque sono stati inseriti dai propri docenti di classe. Successivamente in correlazione con gli studenti sono presenti due tabelle, la prima “SECTION” rappresenta la sezione e classe di uno studente, mentre la seconda tabella “DELAY” contiene tutti i ritardi che sono stati fatti dai vari alunni con varie informazioni. Vi sono ulteriori due tabelle a sé rispettivamente “YEAR” la quale contiene i vari semestri per i vari anni e la tabella “SETTING” che contiene le impostazioni del sito web in formato: nome impostazione e valore.

DA\_REVISIONARE

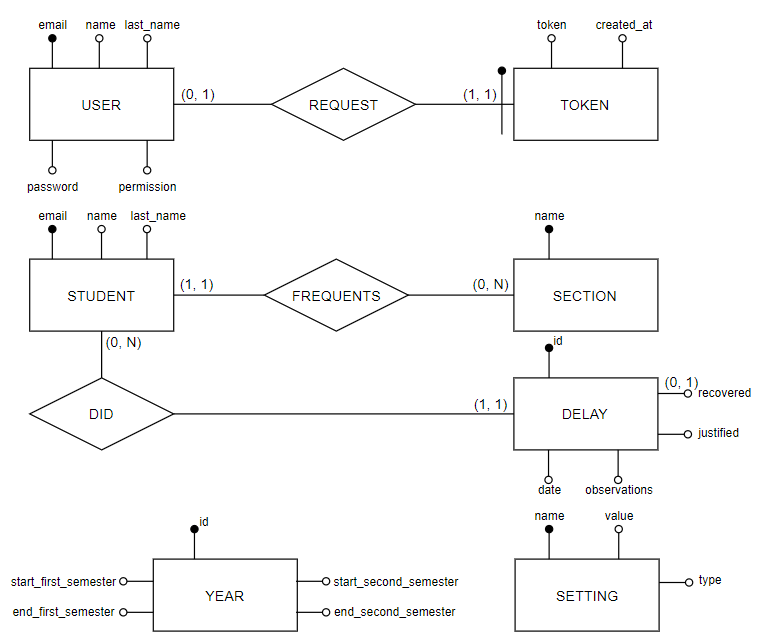


Figura Schema ER banca dati.

### Descrizioni delle tabelle

|  |  |
| --- | --- |
| **USER** | |
| Attributo | Descrizione |
| email | Rappresenta un indirizzo e-mail dell’utente. È un attributo di tipo stringa con limite di 255 caratteri. Non può essere nullo e deve essere univoco.  Esempio: filippo.finke@samtrevano.ch |
| name | Rappresenta il nome di un utente. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Filippo |
| last\_name | Rappresenta il cognome di un utente. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Finke |
| password | Rappresenta la password di un utente. È un attributo di tipo stringa con limite di 60 caratteri. Non può essere nullo. All’interno di questo attributo verrà salvata un hash della password dell’utente.  Esempio: $2y$10$NmiaiLmr3dhUg3ePIExyt.l2KvE7SK6le1UH67QVikBlyBjjTHgVG |
| permission | Rappresenta il permesso di un utente. È un attributo di tipo intero e non può essere nullo. Può contenere i seguenti valori:  1 - Inserimento ritardi.  2 - Visione ritardi.  4 - Creazione PDF.  8 - Amministratore.  Esempio: 8 |

|  |  |
| --- | --- |
| **TOKEN** | |
| Attributo | Descrizione |
| token | Rappresenta un codice che verrà utilizzato per eseguire il recupero della password. Questo codice verrà generato in modo casuale. All’interno dell’attributo verrà salvata un hash in SHA256 del token di recupero password. Il campo è di tipo stringa con un limite di 64 caratteri e non può essere nullo.  Esempio: 4b9eb466ad2615314c8194b1c46e6ef8e910d0b41a682e32de73503883e09b58 |
| created\_at | Rappresenta la data di creazione del codice di recupero password. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-05-12 14:16:59 |

|  |  |
| --- | --- |
| **STUDENT** | |
| Attributo | Descrizione |
| email | Rappresenta un indirizzo e-mail dell’utente. È un attributo di tipo stringa con limite di 255 caratteri. Non può essere nullo e deve essere univoco.  Esempio: filippo.finke@samtrevano.ch |
| name | Rappresenta il nome di un utente. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Filippo |
| last\_name | Rappresenta il cognome di un utente. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Finke |

|  |  |
| --- | --- |
| **SECTION** | |
| Attributo | Descrizione |
| name | Rappresenta il nome della sezione. È un attributo di tipo stringa, ha un limite di 10 caratteri e non può essere nullo.  Esempio: SAM I4AC |

|  |  |
| --- | --- |
| **DELAY** | |
| Attributo | Descrizione |
| id | Rappresenta l’identificatore di un ritardo. È un attributo di tipo intero e viene impostato in modo automatico dall’applicativo. Non può essere nullo e deve essere univoco.  Esempio: 1 |
| recovered | Rappresenta la data di recupero del ritardo. È un attributo di tipo DATE e può essere nullo.  Esempio: 2020-05-12 |
| justified | Flag che indica se il ritardo è giustificato oppure no.  Esempio: 0 |
| date | Rappresenta la data del ritardo. È un attributo di tipo DATE e non può essere nullo.  Esempio: 2020-05-12 |
| observations | Rappresenta le osservazioni correlate con il ritardo. È un attributo di tipo stringa con un limite di 255 valori, può contenere qualsiasi carattere e può essere nullo.  Esempio: Ritardo causa treni. |

|  |  |
| --- | --- |
| **SETTING** | |
| Attributo | Descrizione |
| name | Rappresenta il nome dell’impostazione. È un attributo di tipo stringa, ha un limite di 50 caratteri e non può essere nullo.  Esempio: website\_title |
| value | Rappresenta il valore dell’impostazione. È un attributo di tipo stringa, ha un limite di 255 caratteri e non può essere nullo.  Esempio: Gestione ritardi |
| type | Rappresenta il tipo di dato dell’impostazione.  Esempio: email |

|  |  |
| --- | --- |
| **YEAR** | |
| Attributo | Descrizione |
| id | Rappresenta l’identificatore di un anno. È un attributo di tipo intero e viene impostato in modo automatico dall’applicativo. Non può essere nullo e deve essere univoco.  Esempio: 1 |
| start\_first\_semester | Rappresenta la data di inizio del primo semestre. È un attributo di tipo DATE.  Esempio: 2019-05-10 |
| end\_first\_semester | Rappresenta la data di fine del primo semestre. È un attributo di tipo DATE.  Esempio: 2020-05-11 |
| start\_second\_semester | Rappresenta la data di inizio del primo semestre. È un attributo di tipo DATE.  Esempio: 2020-05-11 |
| end\_second\_semester | Rappresenta la data di fine del primo semestre. È un attributo di tipo DATE.  Esempio: 2020-09-12 |

DA\_REVISIONARE

### Schema logico

Questo è lo schema logico del database:

user(email, name, last\_name, password, permission)

token(email(FK), token, created\_at)

setting(name, value, type)

section(name)

student(email, name, last\_name, section(FK))

delay(id, email(FK), date, observations\*, recovered\*, justified)

year(id, start\_first\_semester, end\_first\_semester, start\_second\_semester, end\_second\_semester)

DA\_REVISIONARE

## Design delle interfacce

### Pagina di accesso

Questo è un mockup della pagina utilizzata per accedere all’applicativo web. Attraverso questa pagina è quindi possibile inserire username e password per poter accedere tramite il database locale.

DA\_REVISIONARE

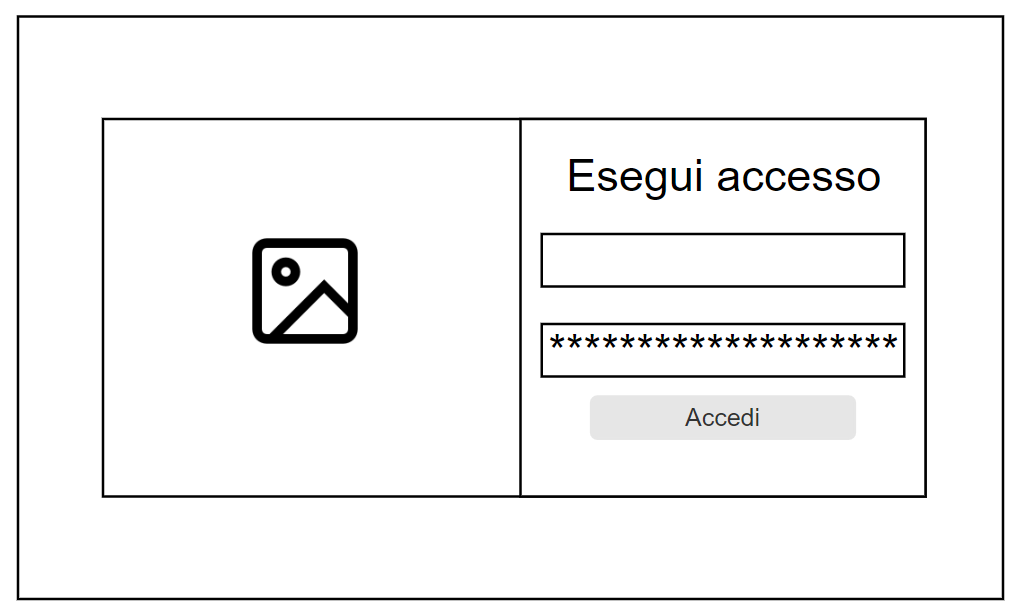


Figura Mockup pagina di accesso.

### Pagina cambio password

Questo è un mockup della pagina utilizzata per cambiare la password del proprio account utente all’interno dell’applicativo web. Questa pagina sarà accessibile solamente attraverso un token di recupero ricevuto tramite posta elettronica.

DA\_REVISIONARE

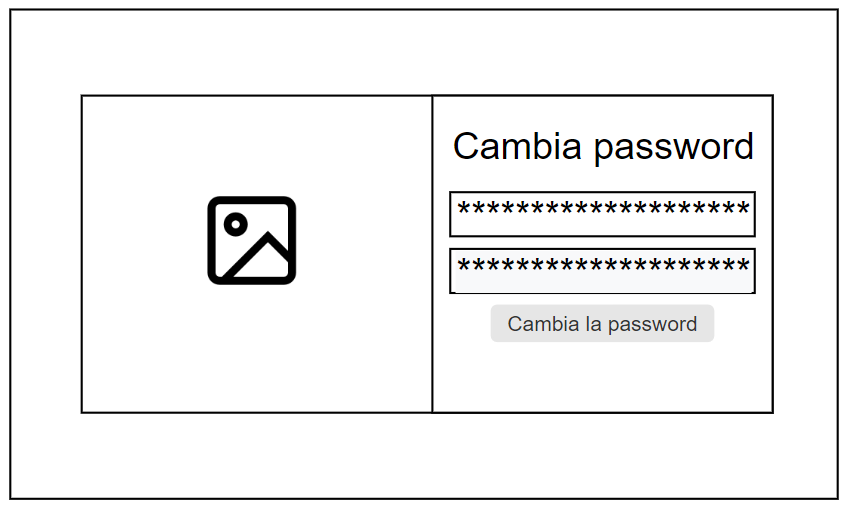


Figura Mockup pagina di cambio password.

### Pagina di recupero password

Questo è un mockup della pagina utilizzata per richiedere una e-mail di recupero password. Inserendo l’email di un account registrato all’interno del sito web verrà inviato un messaggio di posta elettronica contenente un link che permetterà il cambio password.

DA\_REVISIONARE



Figura Mockup pagina di recupero password.

### Pagina di aggiunta ritardi

Questo è un mockup della pagina utilizzata per aggiungere ritardi agli studenti. Nella parte sinistra è presente una barra di navigazione la quale rimarrà anche nel resto delle pagine della dashboard, inoltre nella parte superiore destra verrà mostrato il nome utente di chi ha eseguito l’accesso. Attraverso questa pagina sarà dunque possibile aggiungere ritardi e studenti.

DA\_REVISIONARE

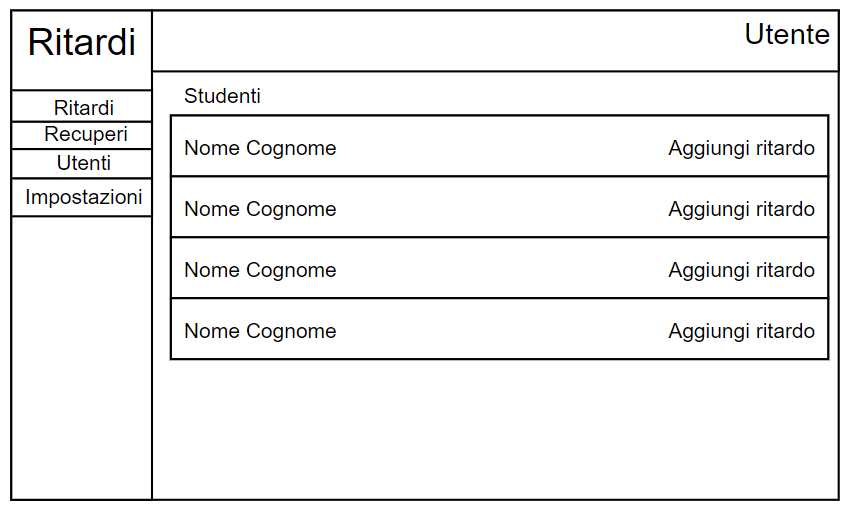


Figura Mockup pagina di aggiunta ritardi.

### Pagina di aggiunta recuperi

Questo è un mockup della pagina utilizzata per aggiungere recuperi di ritardi agli studenti. Questa pagina permetterà dunque di aggiungere i recuperi dei ritardi eseguiti dagli studenti.

DA\_REVISIONARE

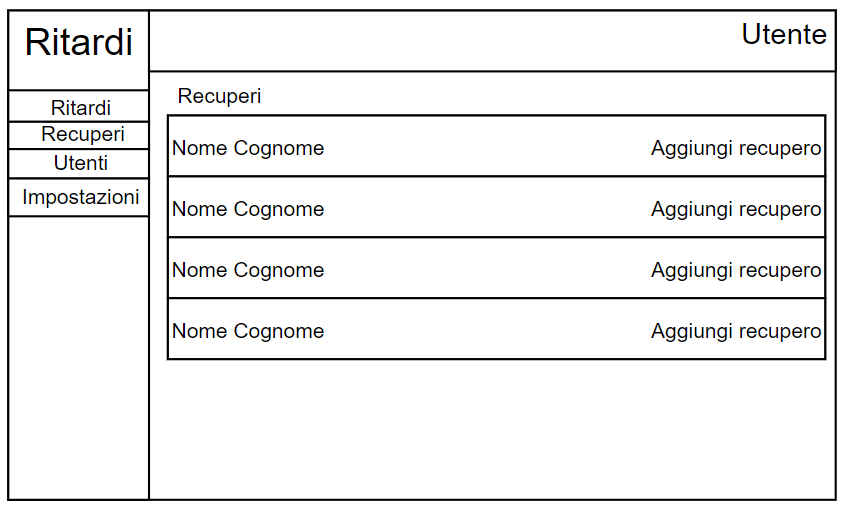


Figura Mockup pagina di aggiunta recuperi.

### Pagina di gestione utenti

Questo è un mockup della pagina utilizzata per gestire gli utenti all’interno dell’applicativo web. Questa pagina sarà accessibile solamente agli amministratori e permetterà di creare nuovi utenti. Alla creazione di un utente esso riceverà un messaggio di posta elettronica contenente un link che permetterà il cambio password.

DA\_REVISIONARE

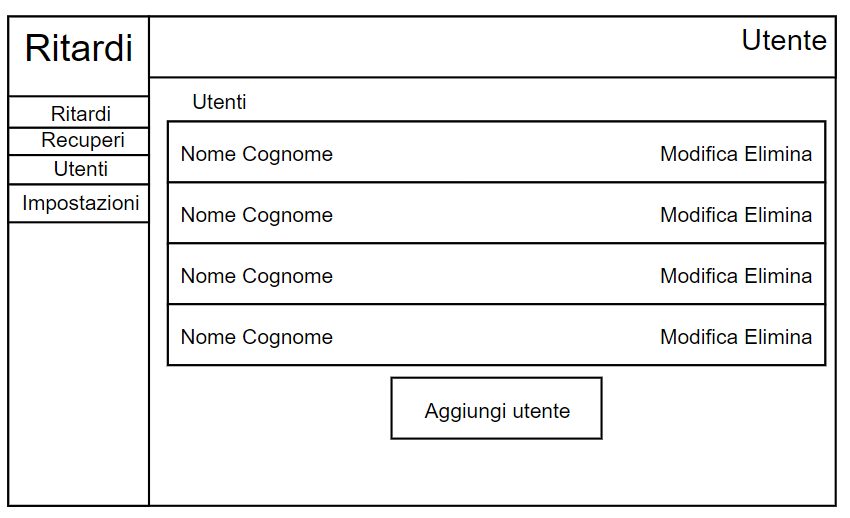


Figura Mockup pagina di gestione utenti.

### Pagina impostazioni

Questo è un mockup della pagina utilizzata per cambiare le impostazioni dell’applicativo web. Anche questa pagina è accessibile solamente da parte degli amministratori dell’applicativo web e permetterà di modificare alcune impostazioni di esso.

DA\_REVISIONARE

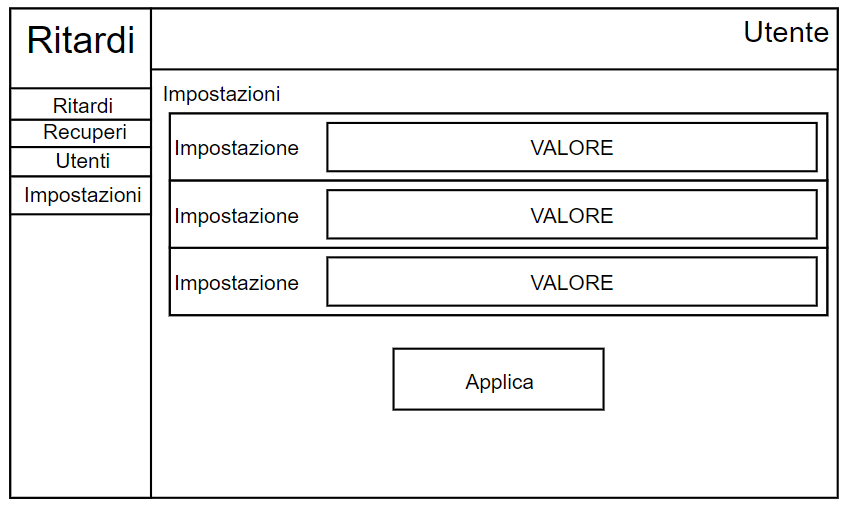


Figura Mockup pagina impostazioni.

# Implementazione

## Gestione versioni

Per gestire i file che compongo l’intero progetto soprattutto il codice di esso ho utilizzato una repository GitLab messa a disposizione dalla scuola su un server interno da parte dei formatori. Questa repository mi permette di tenere traccia di tutti i cambiamenti che ho eseguito ai diversi file del progetto. Inoltre, caricando i file modificati sulla repository del progetto con messaggi adeguati alle modifiche eseguite mi permette di poter tornare in dietro nel codice in ogni momento e di tenere traccia di quanto fatto.

DA\_REVISIONARE

## Gestore di pacchetti

Il progetto utilizza dei pacchetti esterni o librerie per funziona correttamente. Per importare queste librerie esterne è stato utilizzato Composer. Composer è un gestore di pacchetti per il linguaggio di programmazione PHP. Esso permette dunque di scaricare le librerie e di generare i file di caricamento automatico dei vari pacchetti scaricati. Ho dunque creato il file di configurazione chiamato “composer.json” il quale permette di specificare diverse informazioni che andranno utilizzate dal gestore di pacchetti. All’interno del file di configurazione ho dunque impostato come dipendenza richiesta php-rest, la libreria che andrò ad utilizzare per lo sviluppo del sito web.

DA\_REVISIONARE

{

    "name": "filippo/ritardi-web",

    "description": "Gestione Web Ritardi Allievi SAMT",

    "authors": [

        {

            "name": "Filippo Finke",

            "email": "filippo.finke@samtrevano.ch"

        }

    ],

    "minimum-stability": "dev",

    "autoload": {

        "psr-4": {

            "FilippoFinke\\":"src/"

        }

    },

    "require": {

        "filippofinke/php-rest": "dev-master"

    }

}

## Database

### Account di accesso

L’account di accesso al database è stato fornito dal formatore. Esso può creare un database e su di esso ha permessi completi (modifica struttura e dati).

DA\_REVISIONARE

### Implementazione banca dati

Il database è stato implementato seguendo lo schema ER creato nel capitolo di progettazione. Non è stato necessario implementare nessuna tabella ponte aggiuntiva quindi anche il database finale è composto da sette tabelle. Il database inoltre può essere suddiviso in quattro parti differenti. La prima parte comprende le tabelle “USER” e “TOKEN” le quali sono collegate tra di loro attraverso delle chiavi esterne, questa parte viene utilizzata per la gestione degli account che potranno accedere all’applicativo. Successivamente vi sono tre tabelle “STUDENT”, “SECTION” e “DELAY” le quali anch’esse collegate tra di loro attraverso chiavi esterne e vengono utilizzate per salvare i dati riguardanti gli allievi e i ritardi. Oltre a queste sono presenti ulteriori due tabelle completamente distaccate dal resto ovvero “SETTING” e “YEAR” le quali vengono utilizzate per salvare rispettivamente le impostazioni del sito web e i semestri scolastici. La parte che ritengo importante in quanto differente da quanto sviluppato in precedenza è la tabella “SETTING” essa è stata implementata nel seguente modo:

CREATE TABLE setting (

    name VARCHAR(50) PRIMARY KEY,

    value VARCHAR(255) NOT NULL

);

Anche se l’implementazione è molto semplice questa tabella è molto utile perché all’interno di essa è possibile salvare le impostazioni del sito web ed in futuro aggiungerne di nuove nel formato nome impostazione e valore senza dover eseguire grandi modifiche al codice.

DA\_REVISIONARE

### Interrogazione database

Per eseguire la connessione al database e di conseguenza interrogarlo il framework php-rest mi permette di ricavare una connessione ad esso attraverso una classe statica chiamata “Database”. A questa classe è possibile impostare i parametri di connessione quali: server, nome database, username e password. Per impostare questi parametri sono presenti dei metodi setter. Il codice dunque per impostare i parametri e ricavare una connessione è il seguente:

// Imposto indirizzo del server MySQL.

Database::setHost(DB\_HOST);

// Imposto del database da utilizzare.

Database::setDatabase(DB\_NAME);

// Imposto del nome utente per accedere al database.

Database::setUsername(DB\_USERNAME);

// Imposto della password per accedere al database.

Database::setPassword(DB\_PASSWORD);

// Controllo connessione alla banca dati.

try {

    // Successo

    Database::getConnection();

} catch (PDOException $e) {

    // Errore

}

Il metodo getConnection() verrà utilizzato da tutte le classi che dovranno interrogare il database. Questo metodo ritorna una connessione PDO, quindi una volta ricavata si potranno utilizzare tutti i metodi supportati dalla classe PDO di PHP.

La documentazione di PDO si può trovare al seguente indirizzo: <https://www.php.net/manual/en/book.pdo.php>.

DA\_REVISIONARE

## Applicativo web

### Struttura

L’applicativo è stato sviluppato con l’ausilio di php-rest ed il template SB Admin 2. La struttura delle cartelle che compongono il codice è la seguente:

├── assets

├── src

│   ├── Controllers

│   ├── Libs

│   ├── Middlewares

│   ├── Models

│   └── Views

└── vendor

La cartella “assets” contiene tutte le risorse statiche necessarie al funzionamento del sito web, queste risorse sono legate principalmente al frontend ed includono file di stile, JavaScript, immagini e fonts. I file che sono presenti in questa cartella vengono dunque serviti direttamente dal server web. Successivamente la cartella “src” contiene gran parte del codice dell’applicativo, all’interno di questa cartella vi sono altre cartelle le quali si dividono per funzione. La cartella “Controllers” contiene tutte le classi che gestiscono le risposte da dare ai vari percorsi quando vengono visitati da un utente. La cartella “Libs” contiene tutte le classi che vengono utilizzate all’interno del codice come ad esempio la classe di invio di posta elettronica. Successivamente la cartella “Middlewares” contiene tutte le classi che si occupano di stabilire gli accessi ai vari percorsi. All’interno della cartella “Models” invece vengono salvate tutte le classi utilizzate per interfacciarsi con la banca dati dell’applicativo web. Tutta la parte grafica del sito web viene salvata all’interno della cartella “Views”. L’ultima cartella presente chiamata “vendor” contiene tutto il codice che è stato generato in modo automatico dal gestore di pacchetti Composer. L’applicativo web è stato sviluppato utilizzando due pattern architetturali, MVC e REST.

DA\_REVISIONARE

### Model View Controller (MVC)

L’applicativo è in parte strutturato in modo tale da poter utilizzare il pattern MVC, ovvero viene diviso ciò che sono i dati dalle viste e dalla logica.



Figura Schema pattern MVC.

Il pattern MVC è dunque diviso in tre categorie diverse:

* Controller, sono le classi che si occupano di collegare le interfacce grafiche con i dati.
* Model, sono le classi che si occupano di interagire con il database per la gestione dei dati.
* View, sono le interfacce grafiche che vengono mostrate all’utente.

All’interno del progetto “Gestione dei ritardi degli allievi SAMT” le classi “Controller” si occupano di collegare ciò che sono le viste con i dati ricavati dalla banca dati attraverso i “Model”. Le classi “Model” dunque implementano dei metodi per facilitare l’interrogazione del database (come ad esempio: inserimento, aggiornamento ed eliminazione). Mentre le “View” sono semplicemente delle pagine HTML e PHP che verranno mostrate all’utente attraverso i “Controller”.

DA\_REVISIONARE

### Representational State Transfer (REST)

Un’altra parte dell’applicativo è stata sviluppata utilizzando il pattern architetturale REST. Ho deciso di utilizzare REST in modo da semplificare l’implementazione delle chiamate per quando riguarda aggiornamento, inserimento e cancellazione di dati.

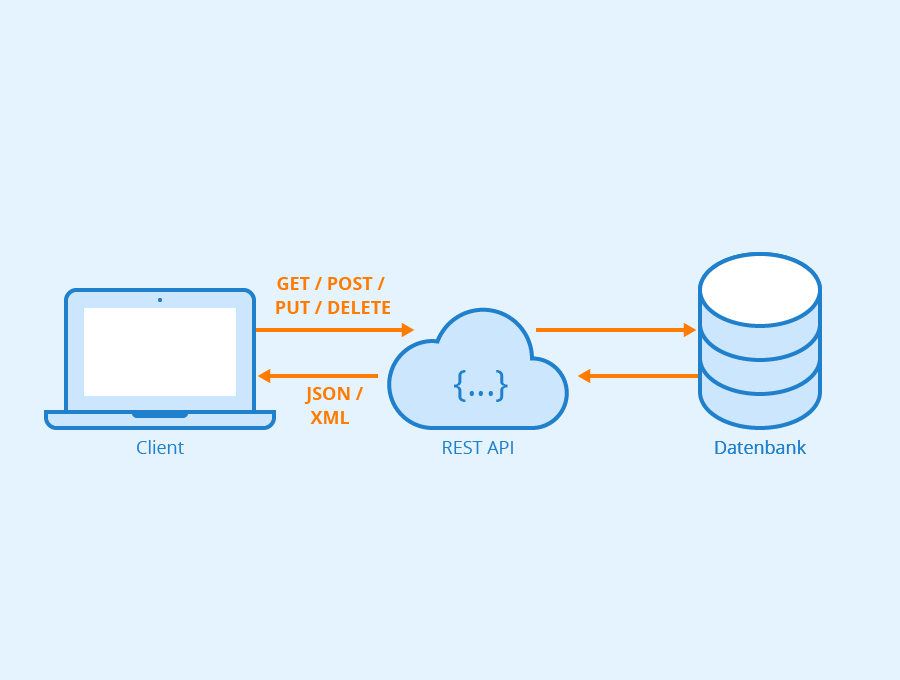


Figura Schema pattern REST.

REST può essere utilizzato per separare completamente il frontend dal backend e quindi di poter supportare diverse interfacce grafiche in quanto solamente le chiamate HTTP alle API REST sono state definite. In questo caso REST viene utilizzato principalmente per rendere le azioni sui dati come ad esempio: inserimento, aggiornamento e cancellazione semplificate attraverso chiamate AJAX da parte di JavaScript, questo inoltre migliora l’esperienza utente in quanto il tutto avviene in background.

DA\_REVISIONARE

### Routing delle richieste

Il framework php-rest mette a disposizione una classe “Router” la quale si occupa di inoltrare le richieste eseguite dagli utenti ai corrispettivi “Controllers”. Per fare questo tutte le richieste che arrivano al webserver devono essere inoltrate ad un file nel quale viene istanziato questo oggetto. All’interno di questo progetto il file di entrata è chiamato “index.php” ed esso contiene tutto quello che riguarda il caricamento dei pacchetti esterni, l’aggiunta di percorsi, caricamento delle impostazioni e altro. Per direzionare tutte le richieste in arrivo al server web verso questo file viene utilizzato il file “.htaccess” il quale permette di stabilire delle regole che permettono di inoltrare le varie richieste in arrivo. Il file .htaccess contiene le seguenti regole:

RewriteEngine On

RewriteBase /

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.+)$ index.php [QSA,L]

La prima istruzione che viene data è quella di attivare il modulo “RewriteEngine” il quale permette di sovrascrivere le richieste. La seconda linea permette di specificare la base che verrà utilizzata per riscrivere le richieste in arrivo. La seconda e terza riga controllano se il file oppure la cartella specificata nella richiesta esista in questo modo è possibile servire file statici senza avere bisogno di passare attraverso il “Router” in PHP. Di conseguenza se le condizioni precedenti non sono state soddisfatte la richiesta viene inoltrata al file “index.php” il quale si occuperà di instradare la richiesta al “Controller” adeguato. L’opzione “QSA” viene utilizzata per specificare di inoltrare anche parametri GET mentre l’opzione “L” dice al webserver di fermarsi se viene eseguita la regola e quindi di non eseguirne altre.

DA\_REVISIONARE

### Configurazione

Il sito web contiene un file di configurazione che permette di specificare alcune impostazioni fondamentali. Attraverso questo file è possibile definire i parametri di connessione al database. All’interno del progetto è presente un file di configurazione di esempio chiamato “config.sample.php” il quale contiene una configurazione di esempio. Questo file è stato creato in modo da guidare l’amministratore alla configurazione del sito web basterà quindi rinominare questo file in “config.php” una volta che il sito web verrà caricato su un hosting. L’applicativo controlla la presenza del file di configurazione nel seguente modo:

// Controllo del file di configurazione.

if (!file\_exists("config.php")) {

    $response = new Response();

    $info = array(

        "title" => "File di configurazione mancante!",

        "message" => "Messaggio di errore."

    );

    $response->render(\_\_DIR\_\_ . "/src/Views/Error/error.php", $info);

    exit;

}

// Includo del file di configurazione dell'applicativo.

require \_\_DIR\_\_ . '/config.php';

Come prima cosa viene dunque controllata la presenza del file di configurazione “config.php”. Se il file non è presente viene creata una nuova risposta caricando la vista “error.php” e passandoci un array contenente le informazioni dell’errore. In questo modo è possibile utilizzare la pagina “error.php” per mostrare tutti gli errori in quanto essa si modifica dinamicamente in base a titolo e messaggio di errore. Mentre se il file di configurazione esiste viene semplicemente caricato.

DA\_REVISIONARE

### Autenticazione

Il codice che si occupa di verificare le credenziali dell’utente all’accesso è presente nel controller di autenticazione all’interno di un metodo chiamato doLogin. Questo metodo si occupa di verificare che la combinazione e-mail e password che è stata passata dall’utente interrogando la banca dati. Il primo controllo che viene eseguito è la validità dell’e-mail, successivamente viene controllato se un utente con l’e-mail specificata è presente all’interno del database. Se l’utente è esistente ne viene verificata la password ed in caso di credenziali corrette viene impostata la sessione come utente autenticato.

public static function doLogin(Request $req, Response $res)

{

    $email = $req->getParam("email");

    $password = $req->getParam("password");

    if (isset($email) && Validator::isValidEmail($email) && isset($password) && Validator::isValidPassword($password)) {

        $user = Users::getByEmail($email);

        if ($user && password\_verify($password, $user["password"])) {

            unset($user["password"]);

            Session::authenticate($user);

            return $res->withStatus(200);

        }

    }

    return $res->withStatus(403);

}

Il metodo inoltre risponde alla richiesta con due codici di stato. Se lo stato della risposta è 403 vuol dire che le credenziali sono errate e che quindi l’accesso è stato negato, mentre se la risposta è 200 vuol dire che le credenziali inserite sono corrette e che di conseguenza l’accesso è stato eseguito con successo.

DA\_REVISIONARE

#### Gestione delle sessioni

Per gestire l’autenticazione dell’applicativo vengono utilizzate delle sessioni le quali mantengono lo stato dell’utente nelle varie pagine del programma. Per velocizzare e semplificare l’utilizzo delle sessioni è stata sviluppata una classe chiamata Session la quale permette di interfacciarsi con i metodi per la gestione delle sessioni del linguaggio in maniera semplificata. Ad esempio, per impostare la sessione di un utente come autenticato è presente il metodo authenticate il quale permette di segnare l’utente come autenticato ed inoltre permette il salvataggio di informazioni aggiuntive alla sessione (come ad esempio: Nome, Cognome, e-mail, etc.). Il codice del metodo authenticate è il seguente:

public static function authenticate($data = null) {

    $\_SESSION["authenticated"] = true;

    if (is\_array($data)) {

        foreach ($data as $key => $value) {

            $\_SESSION[$key] = $value;

        }

    }

}

Per determinare se un utente ha eseguito l’accesso oppure no viene utilizzata la chiave authenticated all’interno della sessione. Inoltre, il metodo in questione accetta un array il quale può essere salvato all’interno della sessione stessa. All’interno della classe Session è inoltre presente un altro metodo che permette di verificare se l’utente corrente ha eseguito l’accesso:

public static function isAuthenticated() {

  return (isset($\_SESSION["authenticated"]) && $\_SESSION["authenticated"] == true);

}

Il codice si occupa dunque di controllare la presenza dell’indice authenticated all’interno della sessione e che esso sia true. Un altro metodo molto importante è logout, questo metodo permette di distruggere la sessione corrente e quindi permette di disconnettere un utente dall’applicativo web, il codice che esegue ciò è il seguente:

public static function logout() {

    session\_unset();

    session\_destroy();

}

In aggiunta al metodo di logout è presente anche un altro metodo utilizzato principalmente dalle interfacce grafiche che permette di ritornare il nome e cognome completo dell’utente corrente:

public static function getFullName()

{

    if (isset($\_SESSION["name"]) && isset($\_SESSION["last\_name"])) {

        return $\_SESSION["name"]." ".$\_SESSION["last\_name"];

    }

    return "Errore";

}

DA\_REVISIONARE

#### Gestione dei permessi

Per verificare i permessi assegnati ad un utente è presente una classe Permission la quale si occupa di tradurre il valore del permesso e di suddividerlo nelle varie attività che un utente può eseguire all’interno del sito. In questa classe sono dunque salvati i valori di ogni singolo permesso:

// Permesso di inserimento dei ritardi.

const INSERT = 1;

// Permesso di visione dei ritardi.

const SELECT = 2;

// Permesso di creazione dei PDF.

const CREATE = 4;

// Permesso di amministratore.

const ADMINISTRATOR = 8;

All’interno della classe inoltre, per ogni permesso è presente un metodo che si occupa di controllare che un determinato bit sia impostato eseguendo una semplice OR tra il permesso da verificare e il permesso dell’utente, ad esempio per determinare se un utente è amministratore è presente il metodo isAdmin:

public static function isAdmin($permission = null)

{

    if ($permission === null) {

        $permission = $\_SESSION["permission"];

    }

    return ($permission & self::ADMINISTRATOR) === self::ADMINISTRATOR;

}

Il metodo si occupa dunque di ricavare il permesso come parametro oppure dalla sessione corrente ed esegue una OR tra il permesso passato e il valore del permesso di amministratore. Per ogni permesso sono presenti dei metodi molto simili a questo dove l’unica differenza è il permesso da controllare. Utilizzando questa classe è dunque possibile distinguere in modo semplice le azioni che potranno essere eseguite da un utente.

DA\_REVISIONARE

#### Gestione percorsi e accessi

Per determinare se un utente può accedere ad un determinato percorso vengono utilizzati dei Middlewares all’interno dell’applicativo. Queste classi si occupano dunque di stabilire se l’utente può eseguire l’accesso ad una determinata pagina oppure negarlo. Ad esempio, il Middleware utilizzato per determinare se un utente ha eseguito l’accesso all’applicativo è il seguente:

class AuthRequired

{

    public function \_\_invoke($request, $response)

    {

        if (!Session::isAuthenticated()) {

            $response->redirect("/login");

            exit;

        }

    }

}

Quando la classe viene invocata viene dunque controllato lo stato della sessione dell’utente corrente. Se l’utente non ha eseguito l’accesso il codice inoltra l’utente alla pagina di login e termina. In questo modo bloccherà l’utente dal visitare pagine riservate solamente ad utenti che hanno eseguito il login. Sono presenti diversi Middlewares all’interno dell’applicativo e la struttura di essi è molto simile. Per creare un percorso attraverso la libreria php-rest è necessario istanziare un oggetto di tipo Router il quale permetterà di registrare tutti i percorsi di esso. Per creare un oggetto Router è sufficiente chiamare il suo costruttore:

// Creo un nuovo oggetto router che si occuperà di smistare le richieste.

$router = new Router();

Successivamente sarà possibile assegnare qualsiasi percorso a questo oggetto ed esso verrà registrato in automatico. Questo oggetto supporta tutte le tipologie di richieste come ad esempio: GET, POST, PUT, DELETE etc. Per registrare un percorso è dunque possibile utilizzare la seguente sintassi:

$router->get('/percorso', 'Classe::metodo’);

$router->post('/percorso', 'Classe::metodo’);

$router->put('/percorso', 'Classe::metodo’);

$router->delete('/percorso', 'Classe::metodo’);

È inoltre possibile raggruppare dei gruppi di percorsi in modo da poterli gestire meglio utilizzando un oggetto di tipo RouteGroup. Ad ogni percorso o gruppo di percorsi è possibile aggiungere dei middlewares prima che il percorso sia visitato oppure dopo attraverso i metodi before e after.

$homeRoutes = new RouteGroup();

$homeRoutes->add(

    $router->get("/", "FilippoFinke\Controllers\Home::index")

)

->before(new AuthRequired()); // Controllo autenticazione.

Ad esempio in questo caso viene creato il gruppo che conterrà tutti i percorsi della dashboard principale. A questo gruppo è stato aggiunto un percorso. Tutte le richieste che verranno servite da questo gruppo eseguiranno come prima cosa il middleware AuthRequired il quale si occuperà di verificare se un utente ha eseguito l’accesso oppure no.

DA\_REVISIONARE

### Invio di e-mail

Per rendere l’invio di messaggio di posta elettronica più semplice ed accessibile possibile ho creato una classe dedicata solamente a questa funzionalità. Questa classe contiene solamente un metodo importante chiamato send il quale si occupa di inviare messaggi di posta elettronica ad un indirizzo e-mail con un soggetto e del contenuto. Il contenuto del metodo è molto semplice in quanto si basa sulla funzione mail di PHP.

public static function send($to, $subject, $message){

    $eol = "\r\n";

    $headers = "From: <" . self::$fromEmail . ">" . $eol;

    $headers .= "Content-Type: text/html; charset=UTF-8" . $eol;

    return mail($to, $subject, $message, $headers);

}

Il metodo si occupa di generare un header conforme allo standard richiesto per inviare messaggi di posta elettronica. In questo header viene impostato il mittente e la tipologia di contenuto del messaggio. Successivamente il metodo si basa sulla funzione mail di php la quale si occupa di collegarsi al server mail specificato nel file di configurazione del linguaggio di programmazione e di inviare il messaggio.

DA\_REVISIONARE

### Validazione dei dati

La validazione dei dati è fondamentale, essa permette di aumentare la sicurezza dell’applicativo e di permettere di non avere incongruenze nei dati all’interno della banca dati. I controlli su di essi vengono effettuati sia lato client (quindi dal browser) e sia lato server (ovvero da PHP). In questo modo anche se il client venisse manipolato vi è un ulteriore controllo che non è possibile oltrepassare. Per la validazione lato server è stata creata una classe Validator la quale contiene tutti i metodi utilizzati per la validazione. Mentre per eseguire la verifica dei dati lato client l’applicativo si appoggia alle funzionalità di validazione di HTML5 e in alcuni casi di JavaScript. Come esempio per eseguire un controllo sul nome inserito da parte dell’utente lato server viene utilizzato il seguente codice:

public static function isValidName($name)

{

    return preg\_match('/^[A-Za-zÀ-ÖØ-öø-ÿ]{1,20}$/', $name);

}

Il metodo in questione accetta un parametro, il quale sarà la stringa da validare. Su questa stringa esegue una espressione regolare la quale controlla che siano presenti solamente dei caratteri dell’alfabeto con i vari accenti ed inoltre ne controlla la lunghezza. Per eseguire lo stesso controllo lato client viene utilizzata la proprietà pattern di un input la quale è messa a disposizione da HTML5:

<input pattern="[A-Za-zÀ-ÖØ-öø-ÿ]{1,20}" name="name" type="text" required>

In questo modo i controlli lato client corrispondo a quelli lato server. All’interno della classe Validator sono presenti svariati metodi di validazione come ad esempio: verifica email, verifica date, verifica numeri e così via.

DA\_REVISIONARE

### Gestione dati ed interrogazione database

L’applicativo web utilizza un database MySQL per lo stoccaggio dei dati. Il sito dunque possiede delle classi specifiche le quali permettono di interrogare questa banca dati per ricavarne i dati da mostrare all’utente. Queste classi sono chiamate Model ed esse si occupano di mettere a disposizione del programmatore dei metodi utili per interrogare specifiche tabelle del database. Per ogni tabella presente nel database è dunque presente anche una classe Model la quale possiede la logica per gestirla. Tutte le classi Model interrogano il database attraverso una connessione ricavata da PDO ed utilizzano solamente Prepared Statements. Vengono utilizzati i Prepared Statements per prevenire attacchi di tipo SQL Injection in quanto questi metodi messi a disposizione da PHP si occupano di validare e verificare le query SQL.

DA\_REVISIONARE

#### Tabella user

La tabella user contiene tutti gli utenti che possono accedere all’applicativo web. Per la gestione di questa tabella è presente una classe model chiamata Users la quale contiene metodi per: ricavare tutti gli utenti registrati, ricavare singoli utenti, inserimento di nuovi utenti, aggiornamento di permessi, eliminazione e cambio password. Il primo metodo, getAll, permette di ricavare tutti gli utenti presenti nel database:

public static function getAll() {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM user";

    try {

        $stm = $pdo->query($query);

        return $stm->fetchAll(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Il secondo metodo invece permette di ricavare tutti i dati di un solo utente attraverso il suo indirizzo email, esso si chiama getByEmail ed accetta come parametro l’e-mail attraverso la quale cercare l’utente:

public static function getByEmail($email) {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM user WHERE email = :email";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':email', $email);

    try {

        $stm->execute();

        return $stm->fetch(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Per inserire un nuovo utente all’interno della banca dati è presente il metodo insert, esso accetta tre parametri: e-mail, nome e cognome. Il metodo in questione si occupa di creare un nuovo utente inserendolo all’interno della banca dati e di utilizzare la classe model Tokens per inviare un codice di attivazione dell’account attraverso un messaggio di posta elettronica. Il codice è il seguente:

public static function insert($email, $name, $lastname) {

    $pdo = Database::getConnection();

    $query = "INSERT INTO user VALUES(:email, :name, :lastname, '', 7)";

    $stm = $pdo->prepare($query);

    $stm->bindValue(':email', strtolower($email));

    $stm->bindValue(':name', ucfirst($name));

    $stm->bindValue(':lastname', ucfirst($lastname));

    try {

        return $stm->execute() && Tokens::sendActivationToken($email);

    } catch (\PDOException $e) {

        throw new Exception("Un utente con questa email esiste già!");

    }

}

Il metodo seguente, chiamato update, permette di aggiornare il permesso di un utente. Il metodo accetta dunque due parametri, l’e-mail dell’utente da modificare e il valore del nuovo permesso da inserire.

public static function update($email, $permission) {

    $pdo = Database::getConnection();

    $query = "UPDATE user SET permission = :permission WHERE email = :email";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':email', $email);

    $stm->bindParam(':permission', $permission);

    try {

        return $stm->execute();

    } catch (\PDOException $e) {

        return false;

    }

}

Per eliminare un utente è presente il metodo delete che attraverso l’e-mail può eliminare degli utenti dal database. Il codice per eliminare un utente è il seguente:

public static function delete($email) {

    $pdo = Database::getConnection();

    $query = "DELETE FROM user WHERE email = :email";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':email', $email);

    try {

        return $stm->execute();

    } catch (\PDOException $e) {

        return false;

    }

}

L’ultimo metodo presente all’interno della classe Users si chiama changePassword e permette di cambiare la password di un utente in base ad un token di recupero ed una nuova password. Il metodo si appoggia dunque alla classe model Tokens per verificare che un token sia valido e che possa essere utilizzato prima di eseguire un aggiornamento alla banca dati, il codice è il seguente:

public static function changePassword($token, $password) {

    if (Validator::isValidPassword($password)) {

        $email = Tokens::useToken($token);

        if ($email) {

            $pdo = Database::getConnection();

            $query = "UPDATE user SET password = :password WHERE email = :email";

            $stm = $pdo->prepare($query);

            $stm->bindParam(':email', $email);

            $stm->bindValue(':password', password\_hash($password, PASSWORD\_DEFAULT));

            try {

                $\_SESSION["login\_email"] = $email;

                return $stm->execute();

            } catch (\PDOException $e) {

            }

        }

    }

    return false;

}

Viene inoltre impostata una variabile di sessione con l’e-mail dell’utente al quale è stata cambiata la password in modo da consigliarla all’accesso.

DA\_REVISIONARE

#### Tabella token

DA\_FARE

#### Tabella student

La tabella student viene utilizzata per salvare tutte le informazioni relative agli studenti. Questa tabella possiede una classe model chiamata Students la quale contiene i seguenti metodi: getAll, getByEmail ed insert. Il metodo getAll viene utilizzato per ricavare tutti gli studenti presenti nel database ed il codice è il seguente:

public static function getAll() {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM student";

    try {

        $stm = $pdo->query($query);

        return $stm->fetchAll(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Successivamente il metodo getByEmail viene utilizzato per ricavare i dati di uno studente attraverso la sua e-mail. Il metodo accetta dunque l’indirizzo e-mail come parametro e se uno studente è presente ne ritorna i dati, l’implementazione è la seguente:

public static function getByEmail($email) {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM student WHERE email = :email";

    try {

        $stm = $pdo->prepare($query);

        $stm->bindParam(':email', $email);

        $stm->execute();

        return $stm->fetch(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

L’ultimo metodo della classe è il metodo insert, esso accetta quattro parametri: e-mail, nome, cognome e sezione scolastica. Attraverso questo metodo è dunque possibile inserire un nuovo studente all’interno del sistema.

public static function insert($email, $name, $lastname, $section) {

    $pdo = Database::getConnection();

    $query = "INSERT INTO student VALUES(:email, :name, :lastname, :section)";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':email', strtolower($email));

    $stm->bindParam(':name', ucfirst($name));

    $stm->bindParam(':lastname', ucfirst($lastname));

    $stm->bindParam(':section', $section);

    try {

        return $stm->execute();

    } catch (\PDOException $e) {

        throw new \Exception("Uno studente con questa email esiste già!");

    }

}

DA\_REVISIONARE

#### Tabella section

La tabella section contiene tutte le sezioni scolastiche presenti all’interno dell’applicativo. La classe model di gestione di questa tabella si chiama Sections e permette di ricavare tutte le sezioni, inserirne di nuove oppure di eliminarne già esistenti. Il primo metodo della classe si chiamata getAll e permette di ricavare tutte le sezioni presenti nella banca dati:

public static function getAll() {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM section";

    try {

        $stm = $pdo->query($query);

        return $stm->fetchAll(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Il secondo metodo, ovvero insert, permette di creare una nuova sezione all’interno dell’applicativo. Questo metodo accetta solamente un parametro il quale sarà il nome della sezione da aggiungere:

public static function insert($name) {

    $pdo = Database::getConnection();

    $query = "INSERT INTO section VALUES(:name)";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':name', $name);

    try {

        return $stm->execute();

    } catch (\PDOException $e) {

        throw new Exception("Un sezione con questo nome esiste già!");

    }

}

L’ultimo metodo presente nella classe è il metodo delete che anch’esso accetta solamente un parametro, ovvero il nome della sezione da eliminare. L’implementazione è la seguente:

public static function delete($name) {

    $pdo = Database::getConnection();

    $query = "DELETE FROM section WHERE name = :name";

    $stm = $pdo->prepare($query);

    $stm->bindParam(':name', $name);

    try {

        return $stm->execute();

    } catch (\PDOException $e) {

        return false;

    }

}

DA\_REVISIONARE

#### Tabella delay

DA\_FARE

#### Tabella year

DA\_FARE

#### Tabella setting

All’interno del database è presente la tabella setting, questa tabella si occupa di gestire tutte le impostazioni del sito web. La classe model di gestione per questa tabella è chiamata Settings e permette di ricavare tutte le impostazioni, ricavare una impostazione dal suo nome, ricavare il valore di una impostazione oppure aggiornarne il valore. Non è possibile aggiungere nuovi valori di impostazioni in quanto esse verranno utilizzate solamente dall’applicativo. Il metodo per ricavare tutte le impostazioni è molto simile alle classi precedenti, il codice è il seguente:

public static function getAll() {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM setting";

    try {

        $stm = $pdo->query($query);

        return $stm->fetchAll(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Successivamente è presente il metodo get il quale accetta come parametro il nome di una impostazione. Questo metodo ritorna tutte le informazioni su di essa, ovvero nome, valore e tipo. Il codice è il seguente:

public static function get($name) {

    $pdo = Database::getConnection();

    $query = "SELECT \* FROM setting WHERE name = :name";

    try {

        $stm = $pdo->prepare($query);

        $stm->bindParam(':name', $name);

        $stm->execute();

        return $stm->fetch(\PDO::FETCH\_ASSOC);

    } catch (\PDOException $e) {

        return false;

    }

}

Il metodo utilizzato più spesso di questa classe è il metodo getValue che attraverso il nome di una impostazione permette di ricavarne il valore, l’implementazione del metodo è la seguente:

public static function getValue($name) {

    $setting = self::get($name);

    if ($setting) {

        return $setting["value"];

    }

    return null;

}

In fine è presente l’ultimo metodo update, esso accetta due parametri: l’impostazione e il nuovo valore da applicare. Questo metodo si occupa di controllare che l’impostazione da aggiornare esista e che la tipologia di dato passata dall’utente da assegnare come impostazione rispecchi la tipologia correlata con l’impostazione all’interno della banca dati. Il codice è il seguente:

public static function update($name, $value) {

    $setting = self::get($name);

    if ($setting) {

        if (call\_user\_func\_array(

array("FilippoFinke\Libs\Validator", "isValid" . $setting["type"]),

array($value)

)) {

            $pdo = Database::getConnection();

            $query = "UPDATE setting SET value = :value WHERE name = :name";

            try {

                $stm = $pdo->prepare($query);

                $stm->bindParam(':value', $value);

                $stm->bindParam(':name', $name);

                return $stm->execute();

            } catch (\PDOException $e) {

                return false;

            }

        } else {

            throw new Exception("Il valore deve essere di tipo: " . $setting["type"]);

        }

    } else {

        throw new Exception("Impostazione inesistente");

    }

}

Il metodo in questione si occupa dunque di eseguire un controllo sulla tipologia del dato chiamando in modo dinamico un metodo di validazione presente nella classe Validator per determinare se il nuovo valore dell’impostazione sarà valido oppure no.

DA\_REVISIONARE

### Creazione PDF

DA\_FARE

### Sicurezza

La sicurezza è molto importante in qualsiasi applicativo, soprattutto gli applicativi che vengono resi disponibili attraverso la rete. Per questo motivo anche per questo sito web sono state applicate delle misure di sicurezza in modo da coprire falle ed attacchi comuni.

DA\_REVISIONARE

#### Interrogazione database

Tutte le richieste che vengono eseguite da parte dell’applicativo al database utilizzano la classe PDO la quale è inclusa in PHP. Questa classe mette a disposizione dei metodi di interrogazione del database sicuri se usati nel modo corretto in quanto si occupano di validare e verificare le query. Utilizzando dunque PDO ed i Prepared Statements permette di proteggere l’applicativo da attacchi di tipo SQL Injection. Attacchi di questo tipo possono essere molto dannosi in quanto permettono ad un utente malintenzionato di eseguire query di sua volontà sulla banca dati dell’applicativo. Di seguito un esempio di come viene utilizzato PDO ed i Prepared Statements:

$pdo = Database::getConnection();

$query = "INSERT INTO section VALUES(:name)";

$stm = $pdo->prepare($query);

$stm->bindParam(':name', $name);

try {

    return $stm->execute();

} catch (\PDOException $e) {

    // Errore

}

Viene utilizzato il metodo bindParam il quale permette di sostituire una variabile all’interno della query da eseguire validando automaticamente il parametrò che andrà sostituito in modo che la query non venga manipolata in nessun caso.

DA\_REVISIONARE

#### Salvataggio dei dati

Come citato in precedenza tutti i dati vengono validati prima di essere salvati all’interno della banca dati. Vengono eseguiti questi controlli sui dati in modo da prevenire attacchi XSS anche chiamati Cross-Site Scripting. Attacchi di tipo XSS permettono ad un utente malintenzionato di far eseguire del codice JavaScript malevolo da parte del sito web in modo involontario. Questo attacco può essere dunque molto pericoloso se il codice malevolo viene salvato all’interno della banca dati e poi distribuito a tutti gli utenti che visitano il sito web. Per ovviare a questo problema vengono dunque eseguiti controlli di validazione su tutti i dati che sono stati passati dagli utenti. In caso un campo sia completamente libero come ad esempio le osservazioni di un ritardo viene utilizzata la funzione htmlspecialchars di PHP la quale permette di rimpiazzare caratteri speciali in entità di HTML facendo in modo che la stringa non venga eseguita. L’utilizzo di questa funzione è molto semplice:

htmlspecialchars($observations)

La funzione in questione accetta dunque del testo e ne ritorna il testo con i caratteri speciali sostituiti da entità di HTML.

DA\_REVISIONARE

#### Salvataggio credenziali

Le credenziali degli utenti vengono salvate all’interno della banca dati dell’applicativo web sotto forma di una hash generata utilizzando l’algoritmo di cifratura bcrypt. Vengono salvate solamente le hash delle password all’interno del database in modo tale che anche se un malintenzionato riesca ad accedere al database in sola lettura non possa risalire alla password di ogni utente e di conseguenza non possa accedere all’applicativo. Il linguaggio PHP mette a disposizione due funzioni utilizzate principalmente per questo genere di cose, la prima funzione è chiamata password\_hash e permette di creare una hash di una stringa di testo mentre la seconda è password\_verify la quale permette di verificare che una stringa passata abbia ed una hash siano uguali. Queste due funzioni vengono dunque utilizzate per salvare e verificare le credenziali degli utenti. Per creare una hash la sintassi è la seguente:

$hash = password\_hash($password, PASSWORD\_DEFAULT);

Viene passato il parametro PASSWORD\_DEFAULT alla funzione il quale indica di utilizzare l’algoritmo più sicuro presente in PHP. Al momento l’algoritmo considerato sicuro è bcrypt. Mentre per verificare se una password corrisponda ad una determinata hash viene utilizzata la funzione password\_verify:

$equals = password\_verify('password', 'hash');

Essa ritorna un valore booleano che determina se le due stringe corrispondono alla stessa cosa. In questo caso l’hash da verificare verrebbe ricavata dalla banca dati. DA\_REVISIONARE

#### Recupero password

Il recupero password viene eseguito attraverso un token di recupero che verrà inviato all’utente tramite un messaggio di posta elettronica. Questo token è composto da una serie di 20 byte casuali. Questo token viene salvato nella tabella token insieme all’indirizzo e-mail dell’utente che ha richiesto il recupero password. All’interno della tabella viene anche salvata la data di creazione del token in modo da poter stabilire una validità di tempo. All’interno del database però viene salvata solamente un hash generata con l’algoritmo sha256 del token in questo. In questo modo seguendo lo stesso principio del salvataggio delle password se un utente ha accesso al database in sola lettura non potrà eseguire il cambio password attraverso il token. Il token viene generato utilizzando la funzione random\_bytes e bin2hex di PHP nel seguente modo:

$token = bin2hex(random\_bytes(20));

Questo token verrà dunque inviato per e-mail all’utente mentre nel database ne verrà salvata l’hash in sha256:

$hash = hash("sha256", $token);

Successivamente per verificare se un token è valido viene controllata semplicemente la presenza di esso nella banca dati e che non sia scaduto attraverso la data di creazione.

DA\_REVISIONARE

### Interfacce grafiche

#### Pagina di accesso

DA\_FARE

#### Pagina di cambio password

DA\_FARE

#### Pagina di recupero password

DA\_FARE

#### Pagina di gestione ritardi

DA\_FARE

#### Pagina di gestione recuperi

DA\_FARE

#### Pagina di gestione utenti

DA\_FARE

#### Pagina impostazioni

DA\_FARE

# Test

## Protocollo di test

DA\_FARE

## Risultati test

DA\_FARE

## Mancanze/limitazioni conosciute

DA\_FARE

# Consuntivo

DA\_FARE

# Conclusioni

DA\_FARE

## Sviluppi futuri

DA\_FARE

## Considerazioni personali

DA\_FARE

# Sitografia

DA\_FARE

# Allegati

DA\_FARE

DA\_REVISIONARE = Completato ma da controllare.

IN\_CORSO = Documentato parzialmente.

DA\_FARE = Da documentare.