Gestione Turni Personale

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi e specifica dei requisiti 4

2.3 Use case 8

2.4 Pianificazione 9

2.5 Analisi dei mezzi 11

2.5.1 Software 11

2.5.2 Hardware 11

3 Progettazione 12

3.1 Design dell’architettura del sistema 12

3.1.1 Model View Controller 12

3.1.2 Struttura delle cartelle 13

3.2 Design dei dati e database 14

3.2.1 Diagramma E/R 14

3.2.2 Descrizione diagramma E/R 15

3.3 Design delle interfacce 16

4 Implementazione 24

4.1 Implementazione database 24

4.2 Applicativo Web 25

5 Test 51

5.1 Protocollo di test 51

5.1 Risultati test 58

5.2 Mancanze/limitazioni conosciute 69

6 Consuntivo 70

7 Conclusioni 71

7.1 Sviluppi futuri 71

7.2 Considerazioni personali 71

8 Glossario 72

9 Sitografia 73

10 Allegati 74

# Introduzione

## Informazioni sul progetto

**Titolo**: Gestione turni personale

**Allievi coinvolti nel progetto**:

* Andrea Curti, [andrea.curti@samtrevano.ch](mailto:andrea.curti@samtrevano.ch)
* Dennis Donofrio, [dennis.donofrio@samtrevano.ch](mailto:dennis.donofrio@samtrevano.ch)
* Gioele Zanetti, [gioele.zanetti@samtrevano.ch](mailto:gioele.zanetti@samtrevano.ch)

**Classe**: I3AC Scuola Arti e Mestieri Trevano, Informatica

**Formatore**: Guido Montalbetti, [guido.montalbetti@samtrevano.ch](mailto:guido.montalbetti@samtrevano.ch)

**Data** **inizio**: 27.01.2022

**Data** **fine**: 05.05.2022

## Abstract

All’inizio di questo progetto ci è stato chiesto di realizzare un applicativo web che permette di facilitare la gestione dei dipendenti. Con questo prodotto è possibile gestire i contrattempi causati da assenze preventivate e non. Ogni volta che un dipendente è assente bisogna organizzarsi per avere un dipendente sostitutivo, ma a volte non è evidente. Forse il datore si dimentica di mandare l’email o la manda troppo tardi. Questo progetto è la soluzione a questo problema, perché permette di modificare l’orario settimanale e manda le emali in automatico. Le email contengono tutte le informazioni necessarie al dipendente per saper cosa fare.

## Scopo

Lo scopo del progetto “Gestione turni personale” è quello di semplificare e velocizzare proprio la gestione degli orari e del personale per i datori di lavoro. L’applicativo permette di modificare e visualizzare velocemente gli orari dei propri dipendenti, con la possibilità di visualizzare un calendario che mostri i turni dei dipendenti e le relative ore di lavoro. Questo permette di evitare sovrapposizioni di orari e gli impiegati hanno la possibilità di visionare il calendario in tempo reale in qualsiasi momento della giornata, così da rimanere aggiornati su relativi cambi di turni.

# Analisi

## Analisi del dominio

L’applicativo verrà utilizzato principalmente da datori di lavoro e i loro dipendenti.

Serve per gestire e visualizzare gli orari e turni dei dipendenti di un negozio o azienda, e permette di risparmiare molto tempo ed evitare sovrapposizioni di orari. I datori hanno la possibilità di cambiare i turni dei propri dipendenti, mentre quest’ultimi possono vedere un calendario che contiene tutti i loro turni con i rispettivi orari.

L’applicativo è semplice da utilizzare con un’interfaccia grafica intuitiva; nella pagina del calendario, la modifica degli orari è fatta tutta con drag and drop.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| ID: REQ-001 | |
| Nome | Creazione database |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessitano i permessi di root. |
| Sotto requisiti | |
| 001 | Installazione MySQL. |

|  |  |
| --- | --- |
| ID: REQ-002 | |
| Nome | Pagina di login |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita di una pagina dove effettuare il login. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si necessita un bottone di conferma. |

|  |  |
| --- | --- |
| ID: REQ-003 | |
| Nome | Pagina home datore di lavoro |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita una pagina per modificare gli orari dei dipendenti. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si dovranno poter spostare i dipendenti nel calendario. |

|  |  |
| --- | --- |
| ID: REQ-004 | |
| Nome | Pagina home dipendenti |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita una pagina per vedere i vari orari di lavoro. |
| Sotto requisiti | |
| 001 | Si necessita una lista per selezionare cosa si vuole vedere. |
| 002 | Si necessita della scelta della data per visualizzare il periodo corretto. |

|  |  |
| --- | --- |
| ID: REQ-005 | |
| Nome | Invio email |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita un sistema per inviare email per notificare quando ci sono cambiamenti di orario. |
| Sotto requisiti | |
| 001 | Si necessita una pagina per cambiare gli orari dei dipendenti. |

|  |  |
| --- | --- |
| ID: REQ-006 | |
| Nome | Pagina statistiche |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita una pagina per le statistiche. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si necessitano dei dipendenti che lavorano. |

|  |  |
| --- | --- |
| ID: REQ-007 | |
| Nome | Algoritmo di popolazione dei negozi. |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita un algoritmo per organizzare in modo automatico gli orari dei dipendenti. |
| Sotto requisiti | |
| 001 | Conoscenza di PHP, HTML e MySQL. |

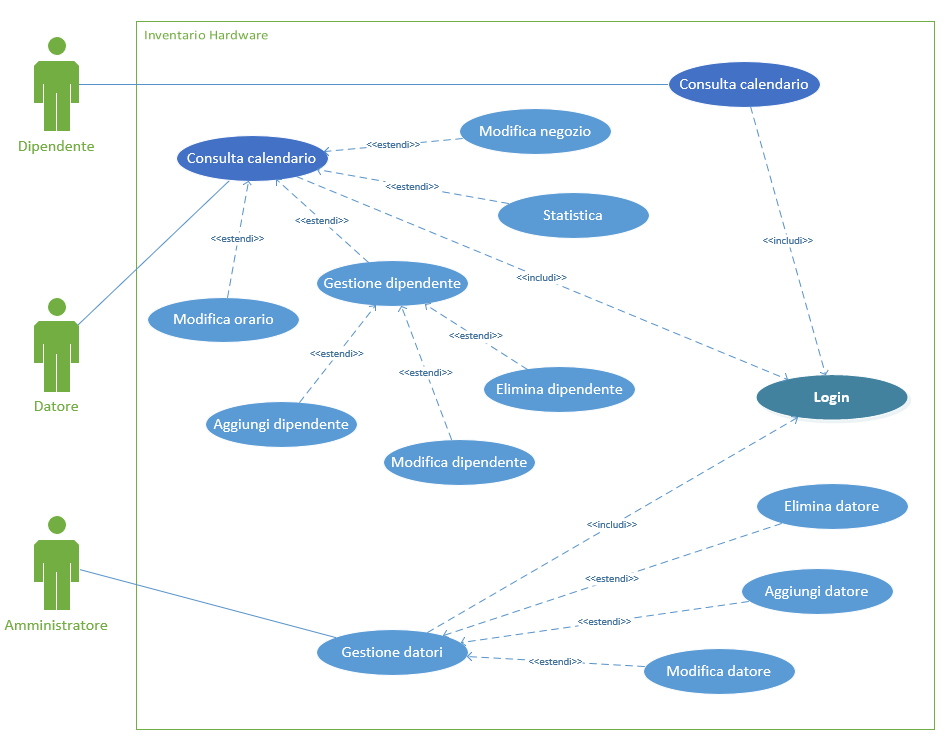
|  |  |
| --- | --- |
| ID: REQ-008 | |
| Nome | Pagine gestione negozi |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessitano delle pagine per la gestione dei negozi. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si necessita una pagina per l’aggiunta di un negozio. |
| 003 | Si necessita una pagina per l’eliminazione di un negozio. |
| 004 | Si necessita una pagina per la modifica di un negozio. |

|  |  |
| --- | --- |
| ID: REQ-009 | |
| Nome | Pagina gestione dipendenti |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessitano delle pagine per la gestione dei dipendenti. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si necessita una pagina per l’aggiunta di un dipendente. |
| 003 | Si necessita una pagina per l’eliminazione di un dipendente. |
| 004 | Si necessita una pagina per la modifica di un dipendente. |

|  |  |
| --- | --- |
| ID: REQ-010 | |
| Nome | Gestione orari e turni |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita pagine per gestire orari e turni |
| Sotto requisiti | |
| 001 | Si necessita una pagina per aggiungere orari |
| 002 | Si necessita di una pagina per modificare gli orari |
| 003 | Si necessita di una pagina per aggiungere un turno |
| 004 | Si necessita di una pagina per eliminare i turni |

|  |  |
| --- | --- |
| ID: REQ-011 | |
| Nome | Calendario |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita di un calendario per visualizzare e inserire i turni dei dipendenti |
| Sotto requisiti | |
| 001 | Si necessita un calendario per impostare e visualizzare i turni |
| 002 | Si necessita diun calendario per visualizzare i propri turni |

## Use case



L’applicativo potrà essere utilizzato da 3 tipi diversi di utenti:

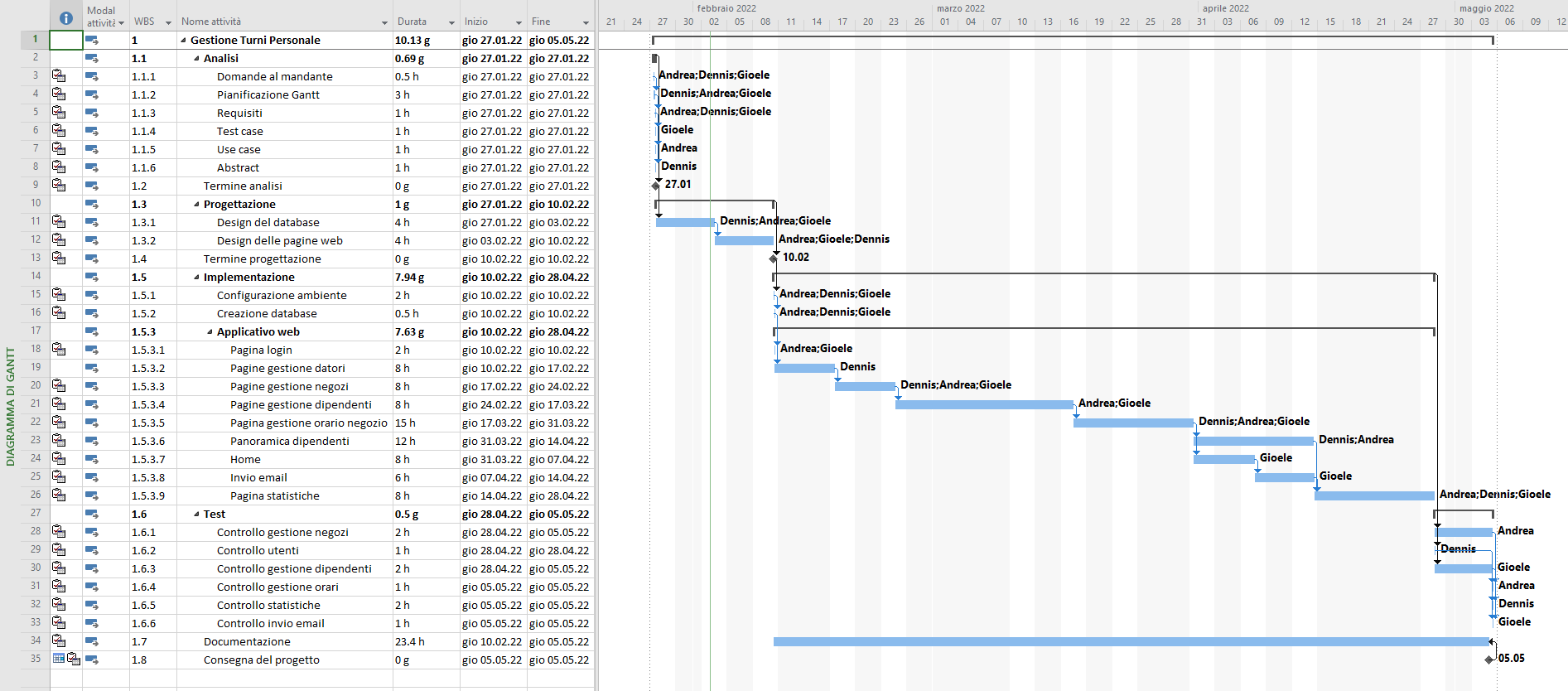
* **Dipendente**: L’unica interazione che questo utente avrà è consultare il calendario per vedere i propri turni di lavoro con i rispettivi orari.
* **Datore**: Questo utente, invece, ha il controllo completo sui propri dipendenti e negozi, infatti può aggiungere, eliminare o modificare sia gli impiegati che i negozi. Un’altra funzione di cui dispone è quella di modificare gli orari e i turni dei dipendenti, e di poter visualizzare una tabella che mostri delle statistiche sugli orari di lavoro di ogni dipendente.
* **Amministratore**: Ha la gestione completa dei datori di lavoro, infatti può aggiungerli, eliminarli o modificarli.

## Pianificazione

Il progetto è stato pianificato suddividendo le attività da svolgere tra le varie persone che lavorano ad esso. Le varie attività sono state suddivise in modo che tutti e 3 le persone che lavorano a questo progetto abbiano sempre qualcosa a cui lavorare, così da non lasciare tempo inutilizzato.

Il Gantt ha 35 righe e contiene varie categorie e attività:

1. **Analisi**: Qui sono raggruppate tutte le attività che riguardano l’analisi del progetto, necessaria per facilitare e velocizzare la progettazione, implementazione e i test.
2. **Progettazione**: Questa categoria contiene la progettazione del database e dell’interfaccia grafica del sito Web, ed un passaggio fondamentale per la buona riuscita del progetto, perché permette di arrivare all’implementazione con le idee chiare su come sviluppare l’applicativo, evitando confusione o rallentamenti.
3. **Implementazione**: Qua iniziano le attività di sviluppo dell’applicativo, ovvero la creazione e implementazione del database e del sito Web.
   1. **Applicativo Web**: È una sottocategoria che contiene tutte le attività riguardanti l’implementazione della parte Web, e si basa sulla progettazione iniziale.
4. **Test**: La parte finale del progetto, ma anche quella più importante, perché serve per controllare che l’applicativo funzioni e soddisfi i criteri e requisiti chiesti dal clienti.



## Analisi dei mezzi

### Software

Programmi e software:

* Apache 2.4
* MySQL 8
* PHP 7
* HTML 5
* CSS 3
* VirtualBox 6.1
* VisualStudio Code 1.58

Librerie:

* Bootstrap 4.6.1 (<https://getbootstrap.com/>)
* FullCalendar 5.10 (<https://fullcalendar.io/>)

### Hardware

Macchina di sviluppo:

* Xeon E3-1240 2 core
* 4GB RAM
* 40GB HDD

Macchina di produzione:

* Host Infomaniak fornito dalla scuola

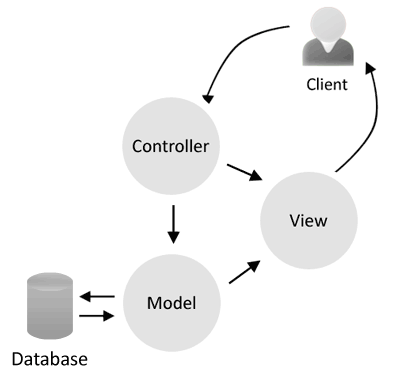
# Progettazione

## Design dell’architettura del sistema

### Model View Controller

Model View Controller (MVC) è una struttura che permette di gestire al meglio tutti i file php. Viene utilizzato nell’ambito web con il linguaggio di scripting PHP. In ogni cartella vengono salvati dei file con compiti differenti.

* **Model:** in questa cartella vengono messe tutte le classi che elaborano i dati e che comunicano con il database
* **View:** in questa cartella vengono messi tutti i file contenenti script html. Non viene messo codice php. Si può usare php in questi file solo per stampare file, quindi è concesso l’utilizzo di “for” e semplici “if”.
* **Controller:** in questa cartella vengono messi tutte le classi che ricevono le richieste da parte del browser. Queste classi elaborano le richieste, prendono i dati necessari dal database grazie ai file nella cartella “model”. Dopo passano gli eventuali dati ai file nella cartella “view” e mandando al browser il file giusto dalla cartella “view”.



### Struttura delle cartelle

Immagine che contiene tavolo

Descrizione generata automaticamente

La struttura delle cartelle utilizzata è composta da 7 cartelle principali:

* **config:** in questa cartella ci sono i file di configurazione come il collegamento al database
* **controller:** in questa cartella vengono inseriti tutti i file contenenti dei controller.
* **libs:** in questa cartella ci sono tutti i file che vengono ripresi più volte nelle varie pagine. Per

evitare di avere lo stesso codice duplicato in vari file, il codice viene scritto il questi file e richiamati all’occorrenza.

* **logs**: in questa cartella ci saranno i file log di ogni evento o errore generato dall’applicativo.
* **models:** in questa cartella ci sono tutti i file contenenti dei models.
* **public:** in questa cartella ci sono tutte le immagini, css, bootstrap ed il codice javascript.
* **views:** in questa cartella ci sono tutti i file contenenti delle views.

## Design dei dati e database

### Diagramma E/R

### Descrizione diagramma E/R

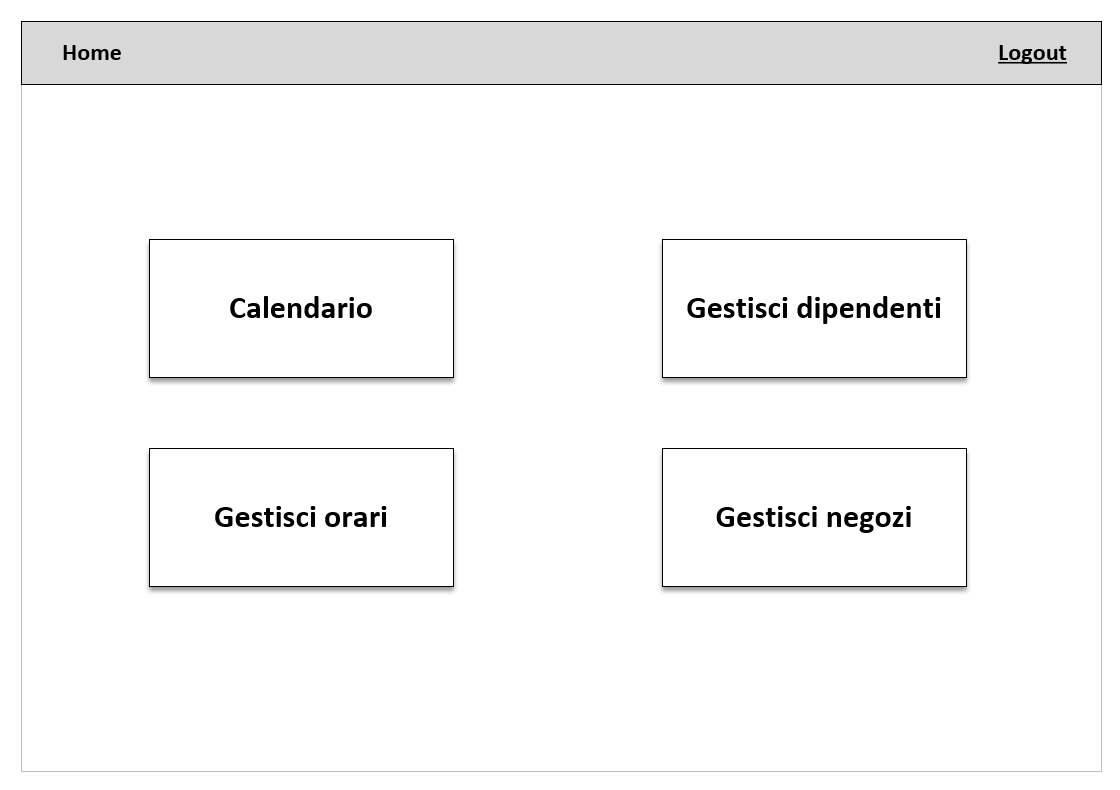
La struttura del database è piuttosto complessa, ed è formata da 11 tabelle, necessarie per il buon salvataggio delle informazioni. Le tabelle sono le seguenti:

1. **Amministratore**: questa tabella non ha relazioni con nessun’altra tabella del database, contiene le informazioni dell’amministratore, ovvero: id, nome, cognome, email e password.
2. **Dipendente**: ci sono tutte le informazioni dei dipendenti, ossia l’id, il nome e cognome, l’email, la password, l’indirizzo di residenza e “archiviato” che serve per capire se l’utente è attivo oppure è “eliminato”, quindi archiviato.
3. **Turno\_lavoro**: in questa tabella vengono salvate le date dei turni di lavoro dei dipendenti.
4. **Orario\_turno**: qua vengono salvate l’ora di inizio e di fine dei vari turni lavorativi.
5. **Negozio**: ci sono tutti dati dei negozi dei datori: l’id univoco, il nome del negozio, l’indirizzo del negozio e l’attributo “archiviato” per sapere se il negozio esiste ancora oppure è stato eliminato e quindi archiviato.
6. **Datore**: la tabella contiene le informazioni sui datori, ovvero l’id, il nome e cognome, l’email, la password, indirizzo di residenza e “archiviato” per conoscere se è ancora attivo oppure no.
7. **Tipo**: è una tabella per sapere la categoria di ogni negozio, e vengono salvate le seguenti informazioni: id univoco, nome e breve descrizione.
8. **Orario**: serve per conoscere l’orario di apertura e chiusura di ogni negozio, e contiene l’id, l’orario di inizio e di fine.
9. **Giorno**: ci sono i giorni della settimana, con l’id e il loro nome.
10. **Usa**: è una tabella relazione che serve per collegare la tabella “Giorno”, la tabella “Orario” e “Negozio” così da sapere l’orario e giorno di apertura dei negozi. Contiene tutte le chiave esterne di queste 3 tabelle
11. **Prevede**: anche questa è una tabella relazione per collegare la tabella “Orario\_turno” e la tabella “Negozio”, così da sapere che orari effettuano i dipendenti del negozio.

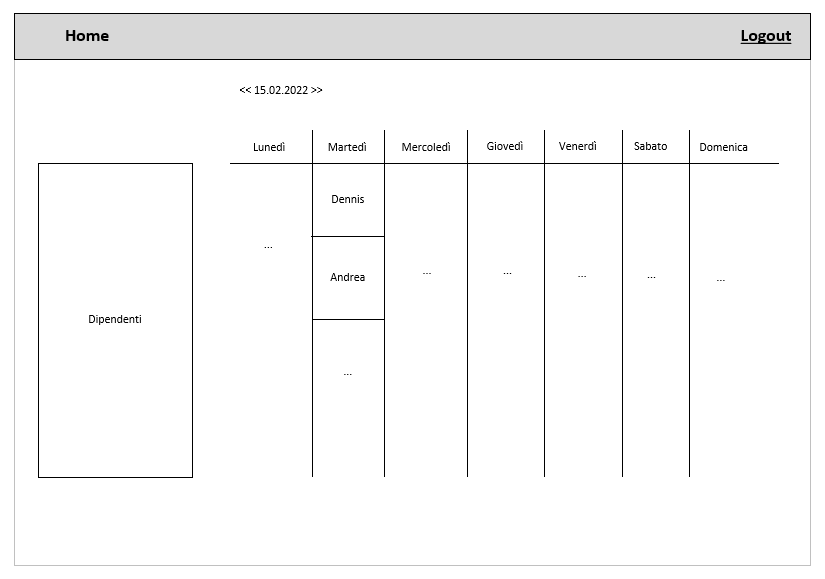
## Design delle interfacce

**Login**

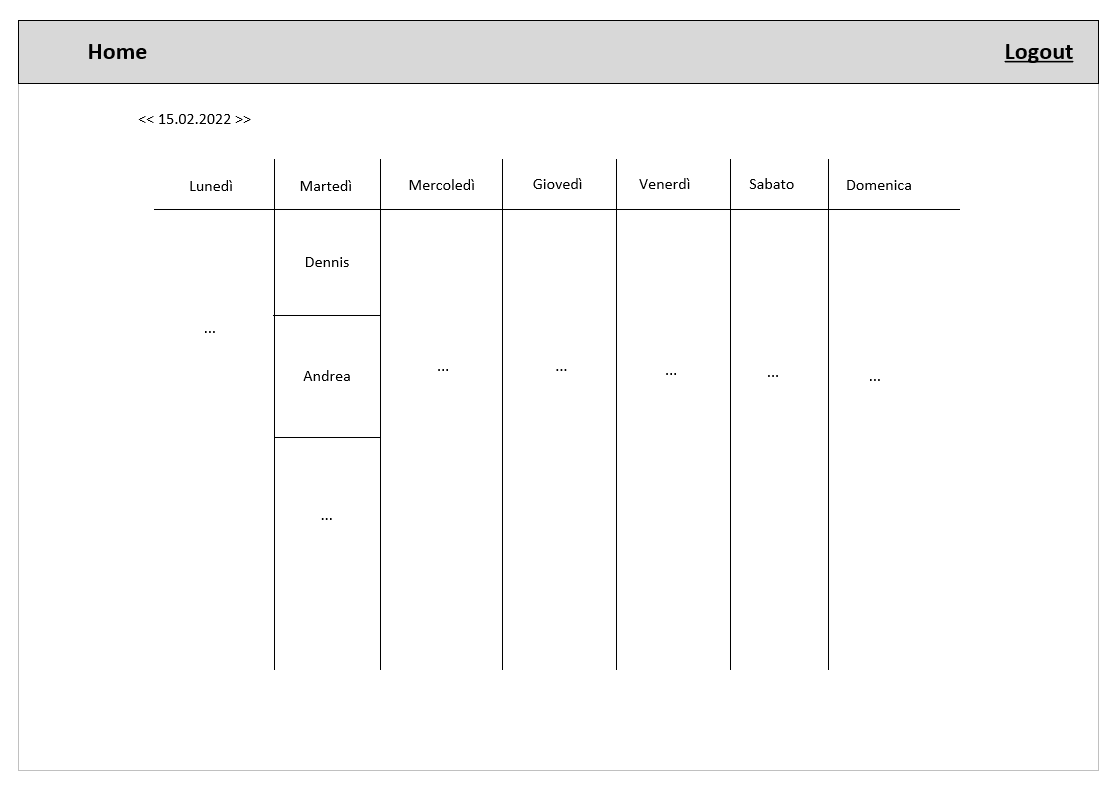
La pagina di login è classica, dove vengono chieste le credenziali di accesso, ed in base al tipo di utente che effettua l’accesso (Amministratore, datore o dipendete), viene aperta una home diversa.

**Home**

Questa è l’interfaccia home per i datori di lavoro, che verrà visualizzata subito dopo aver effettuato il login con un account di questo tipo. La versione dell’utente amministratore contiene solo un bottone per la gestione dei datori, mentre i dipendenti avranno solo la possibilità di visualizzare il calendario degli orari. Per rendere semplice ed intuitivo l’utilizzo dell’applicativo, è stato scelto un design a bottoni.

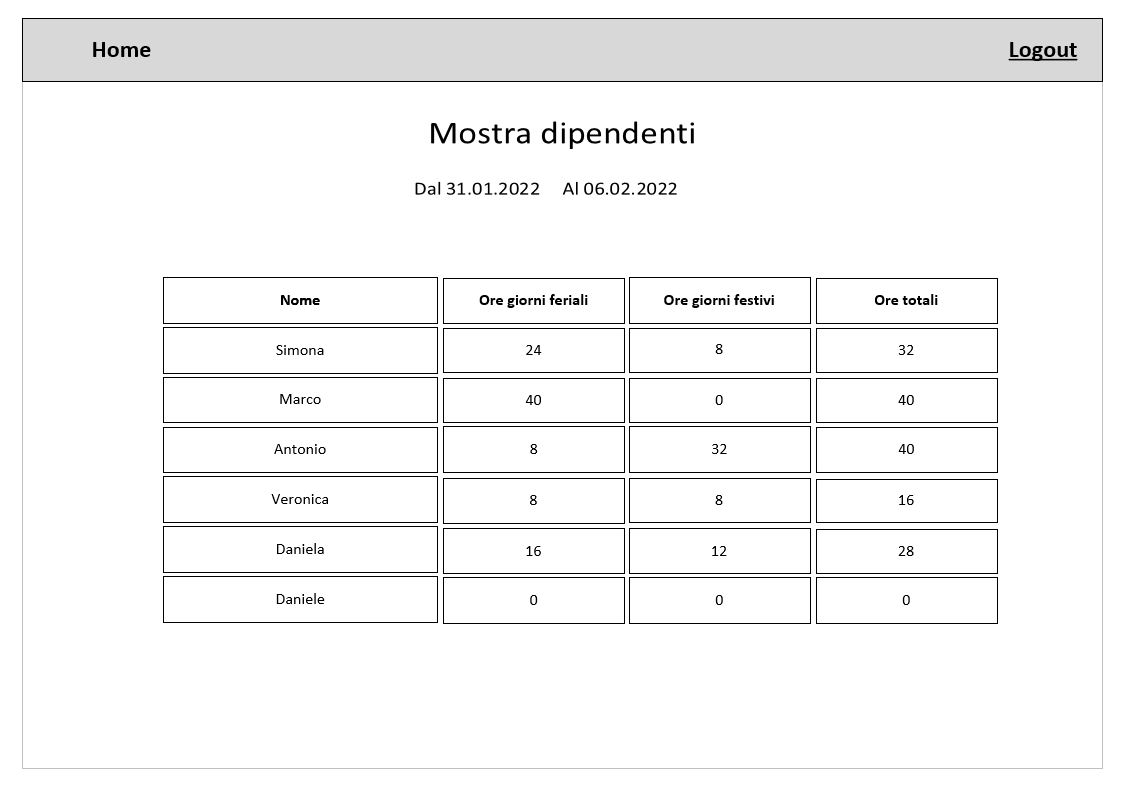
**Modifica turni di lavoro**

Questa pagina potrà essere visualizzata solo dai datori di lavoro e serve per modificare gli orari dei dipendenti di un negozio. I datori potranno trascinare i dipendenti nel calendario, e allungando o accorciando il rettangolo, gli orari di inizio e fine turno cambieranno.

**Calendario**

Questa pagina potrà essere visualizzata dai datori e dai dipendenti di un negozio. È simile alla modifica dei turni di lavoro, con l’unica differenza che qui non possono essere effettuate modifiche, ma solo consultare. In questo modo i dipendenti potranno vedere i propri turni di lavoro in qualsiasi momento su un dispositivo.

**Mostra dipendenti**



La pagina potrà essere consultata solo dai datori per vedere tutte le ore di lavoro che fanno i propri dipendenti. Questo rende molto più facile calcolare lo stipendio di ogni lavoratore, in quanto le ore in giorni festivi sono più retribuite rispetto a quelle in giorni feriali, quindi si può vedere una somma di esse divise tra loro. In alto vengono mostrate le date di intervallo in cui calcolare le ore di lavoro.

**Mostra** **negozi**



Questa pagina mostra tutti i negozi di un datore di lavoro, con le relative informazioni. Ovviamente è consultabile solamente dai datori e nessun altro.

**Aggiungi orari**



Per attribuire un nuovo orario di lavoro ad un negozio si utilizza questa pagina, dove basta selezionare il negozio a cui aggiungere l’orario e poi inserire l’orario di inizio e di fine

**Mostra orari**



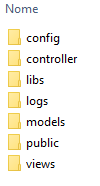
Si possono consultare tutti gli orari disponibili da impostare per i vari negozi.

# Implementazione

## Implementazione database

|  |
| --- |
|  |

## Applicativo Web



La struttura è identica a quella della progettazione. E le cartelle contengono rispettivamente:

**config**: questa cartella contiene il file “config.php”, per configurazioni necessarie, come trovare l’indirizzo host del sito, oppure costanti che vengono utilizzare in tutto il progetto.

**controller**: qui ci sono tutti i controller per le pagine del sito internet. Ognuno di questi controller utilizza un model per effettuare calcoli o modifiche sui dati.

**libs**: vengono raccolte delle classi o librerie utilizzare dai model per effettuare calcoli o operazioni ripetute. Esempio c’è la classe “Hash.php” che serve per calcolare l’hash in sha256 di una password con una salt.

**logs**: in questa cartella ci sono 2 file, uno contiene i log delle attività andate a buon fine, mentre l’altro file contiene tutti gli errori riscontrati durante l’utilizzo dell’applicativo

**models**: si trovano tutti i models richiamati dai controller per svolgere varie funzioni, come l’aggiunta di un componente o utente. Le classi in questa cartella lavorano direttamente sui dati del database.

**public**: in questa cartella sono ci sono 2 sottocartelle: “Calendar”, per i file della libreria del calendario, e “Css”. In quest’ultima cartella si trova a sua volta la sottocartella “Boostrap” che ne contiene altre 3:

* **css:** ci sono le classi di bootstrap con tutti gli stili css
* **icons:** si trovano le icone per i bottoni offerte sempre da Bootstrap
* **js:** questa cartella contiene i file javascript di Bootstrap.

**views**:tutte le pagine contenenti del codice HTML sono raggruppate in questa cartella. Ci sono tutte le pagine del sito web

**config.php**

|  |
| --- |
| $actual\_link = (isset($\_SERVER['HTTPS']) && $\_SERVER['HTTPS'] === 'on' ? "https" : "http") . "://$\_SERVER[HTTP\_HOST]";  $documentRoot = $\_SERVER['DOCUMENT\_ROOT'];  $dir = str\_replace('\\','/',getcwd().'/');  $final = $actual\_link.str\_replace($documentRoot,'',$dir);  define('URL', $final); |

Questo codice serve per trovare il percorso host del sito, così da poter salvare il tutto in una costante “URL” e utilizzarla in tutte le classi per caricare o richiamarne un’altra.

Come prima cosa viene controllato se è https o http, poi viene tornata una stringa $final che contiene il protocollo dell’applicativo con la directory del progetto.

Il codice è stato scritto da Massimo Sartori.

**antiCsScript.php**

|  |
| --- |
| public static function checkAntiScript($text){      if(strpos($text, 'script')){          $text = str\_replace("script", "", $text);      }      return $text;  } |

Questo metodo serve per controllare se una stringa contiene parti di script che potrebbero essere eseguite involontariamente, in caso ci sia, viene eliminata quella parte.

|  |
| --- |
| public static function check($data) {      $data = trim($data);      $data = stripslashes($data);      $data = htmlspecialchars($data);      return AntiCsScript::checkAntiScript($data);  } |

Il metodo “check” server per “sanificare” una stringa, rimuovendo gli spazi vuoti all’inizio e alla fine, rimuove caratteri speciali e richiama il metodo per controllare che non contenga parti di script.

**Connection.php**

|  |
| --- |
| <?php      $servername = "localhost";      $username = "root";      $password = "root";      $dbname = "gestione\_personale";      $conn = new mysqli($servername, $username, $password, $dbname);      if($conn->connect\_error){          die("Connection failed: ".$conn->connect\_error);      }  ?> |

Questa classe viene richiamata da ogni metodo che deve interrogare o comunicare col database;

infatti, stabilisce una connessione con il database, effettuando l’accesso.

**Controller.php**

|  |
| --- |
| public function locate($path){      header("Location: " . URL . $path);  } |

Il metodo “locate” serve per caricare la pagina index del controller passato come argomento.

|  |
| --- |
| public function isLogged(){      if(!empty($\_SESSION['id'])){          return $\_SESSION['role'];      }else{          return -1;      }  } |

Questa funzione serve per controllare se un utente è loggato, se lo è torna il suo ruolo, ovvero dipendente, datore o amministratore.

**email.php**

|  |
| --- |
| public function isOldEmail($id, $isDatore){      require 'application/libs/connection.php';      if($isDatore == true){          $sql = $conn->prepare("SELECT \* FROM datore WHERE id=? AND archiviato=0");          $sql->bind\_param("s", $id);      }else{          $sql = $conn->prepare("SELECT \* FROM dipendente WHERE id=? AND archiviato=0");          $sql->bind\_param("s", $id);      }      $sql->execute();      $result = $sql->get\_result();      $finEm = $result->fetch\_assoc()['email'];      if($finEm == $this->fullEmail){          return true;      }      return false;  } |

Questo metodo serve per controllare se la nuova email di un utente che verrà modificata è identica a quella precedente oppure è diversa.

|  |
| --- |
| public function checkEmailsInDB(){      require 'application/libs/connection.php';      $sql1 = $conn->prepare("SELECT \* FROM dipendente WHERE email=? AND archiviato=0");      $sql1->bind\_param("s", $this->fullEmail);      $sql2 = $conn->prepare("SELECT \* FROM datore WHERE email=? AND archiviato=0");      $sql2->bind\_param("s", $this->fullEmail);      $sql3 = $conn->prepare("SELECT \* FROM amministratore WHERE email=?");      $sql3->bind\_param("s", $this->fullEmail);      for($i=1;$i<=3;$i++){          $currentQuery = 'sql' . $i;          $$currentQuery->execute();          $result = $$currentQuery->get\_result();          if($result){              if ($result->num\_rows > 0) {                  return false;              }          }          $$currentQuery->close();      }      return true;  } |

Questa funzione ritorna se una email è già in utilizzo all’interno del database, controllando tutte e 3 le tabelle degli utenti, ovvero dipendente, datore ed amministratore. In questo modo si assicura che non ci siano utenti con email uguali, a meno che uno dei 2 utenti non sia archiviato.

**hash.php**

|  |
| --- |
| function doHash($salt){  $this->hashedText = hash('sha256', $salt . hash('sha256', $this->plainText));  } |

Viene richiamato qualora si volesse fare l’hash di una password in sha256. C’è bisogno di passare come parametro la salt, ovvero l’email. Questo serve per creare una password sicura e non salvare sul database i dati in chiaro.

**log.php**

|  |
| --- |
| public static function writeLog($msg){      if($puntatore = fopen('application/logs/log.log', "a")){          $user = isset($\_SESSION['id']) ? $\_SESSION['id'] : "undefined";          $role = isset($\_SESSION['roleType']) ? $\_SESSION['roleType'] : "no role";          $str = date("Y/m/d H:i:s"). " role=" . $role . ", user\_id=" . $user . ": " . $msg;          fwrite($puntatore, $str . PHP\_EOL);      }  } |

Il metodo scrive all’interno di un file log e serve per registrare qualsiasi modifica del database, insieme all’orario della modifica, il ruolo dell’utente che ha effettuato la modifica insieme al suo id, e il messaggio passato come parametro. Esiste anche il metodo writeErrorLog() che è identico a questo, ma salva gli errori in un altro file.

**password.php**

|  |
| --- |
| public function isValid(){      $pass = $this->password;      $uppercase = preg\_match('@[A-Z]@', $pass);      $lowercase = preg\_match('@[a-z]@', $pass);      $number = preg\_match('@[0-9]@', $pass);      $specialChars = preg\_match('@[^\w]@', $pass);      if(!$uppercase || !$lowercase || !$number          || !$specialChars || strlen($pass) < 8) {              return false;      }      return true;  } |

Anche in questa classe c’è il metodo isValid(), che controlla che la password rispetti dei criteri: abbia una maiuscola, una minuscola, un numero, un carattere speciale e che sia lunga almeno 8 caratteri.

**view.php**

|  |
| --- |
| public function render($name, $onlyIncludeBody = false, $data = array()){      if($onlyIncludeBody){          require "application/views/" . $name;      }else{          require 'application/views/header.php';          require "application/views/" . $name;          require 'application/views/footer.php';      }  } |

La funzione “render” serve per caricare una view col nome passato come argomento. C’è la possibilità di scegliere se includere l’header ed il footer, in più anche la possibilità di inserire dei dati tramite l’array “data”.

**calendario.php**

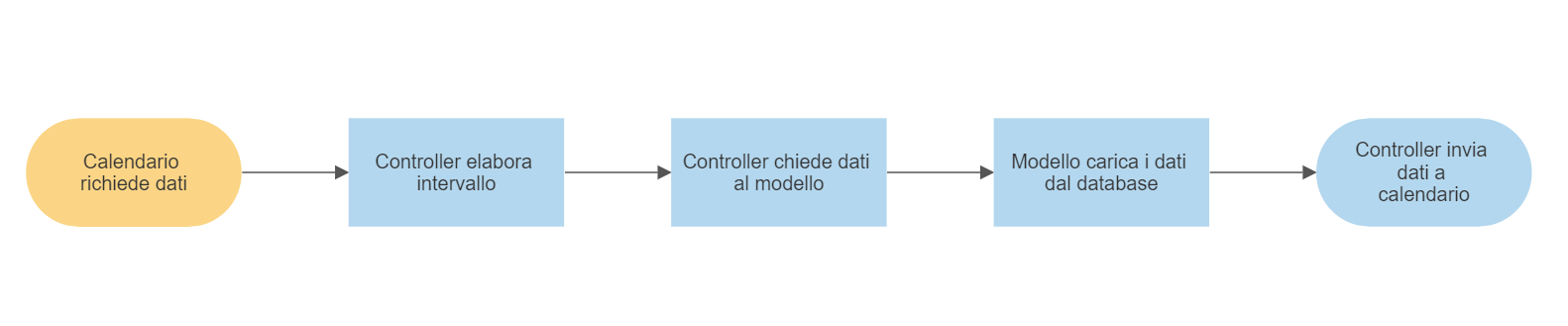
|  |
| --- |
| public function index(){      if($this->isLogged() == 2){          parent::getModel('negozio\_model.php');          $model = new NegozioModel();          $this->view->render('calendario/negozio.php', false, array('negozi' => $model->ottieniNegozi()));      }else{          $this->view->render('login/index.php');      }  } |

Il metodo “index” è presente in tutti i controller e serve per richiamare la view principale di ogni controller. Tutti i metodi di tutti i controller, tranne per login, possono essere richiamati da utenti loggati e con di una specifica categoria (dipendente, datore o amministratore). **Questo metodo è presente in tutti i controller per controllare l’accesso.**

Calendario è un controller che si occupa di prelevare gli eventi dei lavoratori nel database e visualizzarli all’interno del calendario. Per fare questo, la libreria FullCalendar visita un url mettendo nel get l’inizio e la fine del periodo visualizzato nel calendario.

|  |
| --- |
| public function ottieniEventi(){      if($this->isLogged() == 1 || $this->isLogged() == 2){          parent::getModel('orario\_model.php');          $inizio = $\_GET['start'];          $fine = $\_GET['end'];          $model = new OrarioModel();          $json = json\_encode($model->ottieniEventiInRange($inizio, $fine));          echo $json;      }else{          $this->view->render('login/index.php');      }  } |

Come si può vedere da questo codice, il controller prende l’intervallo dal get e delega il compito di trovare gli eventi al model. Il risultato è poi stampato sulla pagina in formato json così che il calendario possa interpretare i vari eventi.



|  |
| --- |
| public function salva(){      if($this->isLogged() == 2){          $data = json\_decode($\_POST['data'], true);          $range = json\_decode($data['range'], true);          $events = json\_decode($data['events'], true);          parent::getModel('orario\_model.php');          $model = new OrarioModel();      }else{          $this->view->render('login/index.php');      }  } |

Tramite la funzione salva, è possibile salvare i dati all’interno del database. Anche in questo caso, il compito è delegato al model.

**dipendente.php**

|  |
| --- |
| public function aggiungiDipendente(){      if($this->isLogged() == 2){          parent::getModel("dipendente\_model.php");          $model = new DipendenteModel();          try{              $em = $model->aggiungiDipendente();              Log::writeLog("Nuovo dipendente ".$em." aggiunto");              $this->locate('dipendente');          }catch(Exception $e){              Log::writeErrorLog("Errore nell'aggiunta di un dipendete: ". $e->getMessage());              $this->view->render('gestioneDipendenti/aggiungiDipendente.php',  false, array('error' => $e->getMessage()));          }      }else{          $this->view->render('login/index.php');      }  } |

Dipendente è il controller che si occupa di gestire i dipendenti. Ha quindi metodi per aggiungere, rimuovere, modificare e mostrare tutti i dipendenti di un certo datore e le rispettive pagine di inserimento.

I dipendenti sono di un datore dal momento che vengono immessi da questo nel sistema.

Questo controller è accessibile soltanto dai datori, in quanto questi devono aggiungere i loro dipendenti.

Il codice qui sopra serve per aggiungere un dipendente al database, richiamando i vari metodi del model corrispettivo. In base alla riuscita dell’operazione, viene salvato sul log il risultato, ovvero se è stato aggiunto oppure se è avvenuto un errore.

**gestioneDatori.php**

|  |
| --- |
| public function mostra(){      if($this->isLogged() == 3){          require 'application/models/datoreModel.php';          $model = new DatoreModel();          $this->view->data = $model->ottieniTuttiDatoriCompleti();          $this->view->template = array("id", "nome", "cognome", "email", "indirizzo");          $this->view->render("gestioneDatori/mostra.php");      }else{          $this->view->render('login/index.php');      }  } |

Questo controller serve per gestire i datori dell’applicazione. Questo controller è accessibile soltanto all’/agli amministratore/i, che li devono registrare manualmente.

Anche in questo caso i metodi di questo controller sono per aggiungere, modificare, eliminare e mostrare tutti i datori all’interno del database. Quest’ultima operazione viene eseguita tramite il codice presente nel riquadro sopra. Per fare ciò richiama un metodo per ottenere le informazioni di tutti i datori, e poi li manda alla view e la richiama in seguito.

**gestioneOrari.php**

Questo controller si occupa di gestire gli orari di apertura e chiusura dei negozi. Anche in questo caso ci sono metodi di aggiunta, rimozione, modifica e un metodo per mostrare gli orari esistenti.

**gestioneTipi.php**

Questa classe permette di gestire i tipi dei negozi. Per esempio, un negozio potrebbe essere di tipo “alimentari”. Gli utenti che hanno accesso a questo controllo sono soltanto gli amministratori, che possono aggiungere e mostrare questi tipi.

**gestioneTurni.php**

Questo controller permette di inserire, modificare, eliminare e mostrare tutti i turni di lavoro nel database. Questo controller è accessibile dai datori in modo che possano aggiungere i turni di lavoro dei propri negozi.

**home.php**

Home è il controller che gestisce le pagine home. Si occupa di mostrare le pagine giuste a dipendenza del ruolo. Un amministratore vedrà i tasti per la gestione dei datori e dei tipi, mentre un dipendente vedrà il calendario con i suoi turni di lavoro.

**login.php**

|  |
| --- |
| public function loginUser(){      require 'application/models/login\_model.php';      if($\_SERVER["REQUEST\_METHOD"] == "POST"){          $user = new LoginClass($\_POST["email"],              $\_POST["password"]);          if($user->doLogin()){              $this->view->locate('home');          }else{              $this->view->render('login/index.php', false, array('error' => "Email o password non valida"));          }      }  } |

Questo controller si occupa di permettere il login e salvare le varie informazioni dell’utente loggato in variabili di sessione.

Come si può vedere, viene istanziato il model che provvede a controllare nel database se le credenziali sono valide. Siccome ci sono tre tabelle che contengono utenti, il model deve controllare in ognuna di queste. Ma di questo si parlerà meglio quando verrà descritto il controller in questione.

**mail.php**

Mail si occupa di gestire l’invio delle mail. È un altro di quei controller che non ha una pagina vera e propria, ma viene chiamato una volta che vengono aggiornati i turni di lavoro. Ha dei metodi utili a inviare mail ai i dipendenti di un certo negozio e per visualizzare il template della mail.

**negozio.php**

|  |
| --- |
| public function salvaOrario(){      if($this->isLogged() == 2){          parent::getModel('negozio\_model.php');          $model = new NegozioModel();          $model->salva($\_POST);          $this->locate('negozio');      }else{          $this->view->render('login/index.php');      }  } |

Questo controller è stato creato per gestire i negozi. Ha metodi per inserire, modificare, eliminare e mostrare i negozi nel database ma non solo.

Visto che gli orari di apertura e chiusura sono inerenti ad un certo negozio, abbiamo deciso di includere anche un metodo per poter salvare gli orari che il datore sceglie.

**pianifica.php**

|  |
| --- |
| public function pianificaOrario(){      parent::getModel('pianifica\_model.php');      $model = new PianificaModel();      $negozio = AntiCsScript::check($\_POST['negozio']);      $inizio = AntiCsScript::check($\_POST['inizio']);      $fine = AntiCsScript::check($\_POST['fine']);      $result = $model->pianifica($negozio, $inizio, $fine);      if(is\_bool($result)){          Log::writeLog("Pianificazione avvenuta per negozio con id $negozio da $inizio a $fine");          parent::locate("/");      }else{          Log::writeErrorLog("Errori! $result");      }  } |

Questo controller serve per pianificare l’orario di lavoro di un negozio automaticamente. Come si può vedere, vengono presi i dati del negozio, del giorno di inizio e di fine e della pianificazione e vengono dati in pasto al model, che si occuperà poi di pianificare l’orario. Se qualche dipendente dovesse essere occupato e quindi in un certo turno nel negozio non ci potessero essere almeno due dipendenti, la pianificazione fallisce.

**statistiche.php**

Statistiche serve per generare le statistiche di un certo negozio in un certo lasso di tempo. Le statistiche comprendono le ore totali dei lavoratori, quelle nei giorni feriali (che non sono quelli di vacanza anche se la parola ferie è simile) e i giorni festivi. I metodi della classe servono quindi per stampare i dati di statistica e convertire l’intervallo di tempo passato dall’utente.

**datoreModel.php**

|  |
| --- |
| function modificaDatore(){      require 'application/libs/connection.php';      if(!empty($\_POST['id']) && !empty($\_POST['nome']) && !empty($\_POST['cognome'])      && !empty($\_POST['email']) && !empty($\_POST['pass1'])      && !empty($\_POST['pass2']) && !empty($\_POST['indirizzo'])){          require 'application/libs/hash.php';          require 'application/libs/email.php';          require 'application/libs/password.php';          $this->id =AntiCsScript::check($\_POST['id']);          $this->nome = AntiCsScript::check($\_POST['nome']);          $this->cognome = AntiCsScript::check($\_POST['cognome']);          $this->email = AntiCsScript::check($\_POST['email']);          $this->pass1 = AntiCsScript::check($\_POST['pass1']);          $this->pass2 = AntiCsScript::check($\_POST['pass2']);          $this->indirizzo = AntiCsScript::check($\_POST['indirizzo']);          $emailUser = new Email($this->email);          if(!$emailUser->isOldEmail($this->id, TRUE)){              $emailUser->isValid();          }          if($this->pass1 == $this->pass2){              $passUser = new Password($this->pass1);              if($passUser->isValid()){                  $hp = new Hash($this->pass1);                  $hp->doHash($this->email);                  $this->hash\_password = $hp->getHashed();                  $sql = $conn->prepare("UPDATE datore set nome=?, cognome=?, email=?, hash\_password=?, indirizzo=? WHERE id=?");                  $sql->bind\_param("sssssi", $this->nome, $this->cognome, $this->email, $this->hash\_password, $this->indirizzo, $this->id);                  $result = $sql->execute();                  return $this->email;              }else{                  throw new Exception("La password deve contenere almeno:<br>- 8 Caratteri<br>-1 Maiuscola<br>-1 Minuscola<br>-1 Cifra<br>-1 Carattere speciale");              }          }else{              throw new Exception("Le due password non corrispondono");          }      }else{          throw new Exception("Completare tutti i campi");      }  } |

La classe “datoreModel” serve per fare le richieste al database inerenti alla gestione dei datori.

Questo codice serve per modificare un datore: come prima cosa viene controllato che tutti i dati passati non siano vuoti, poi le varie informazioni vengono controllate con la libreria “AntiCsScript”,

In seguito, viene verificato se la nuova e-mail è identica a quella precedente oppure no e come ultimo controllo c’è quello della password, ovvero se supera tutti i vari requisiti e se è uguale alla password di conferma. Se tutte le verifiche vanno a buon fine, viene modificato il datore con le nuove informazioni nel database.

**dipendente\_model.php**

|  |
| --- |
| public function aggiungiDipendente(){      require 'application/libs/connection.php';      if(!empty($\_POST['nome']) && !empty($\_POST['cognome']) && !empty($\_POST['email'])      && !empty($\_POST['password1'])&&!empty($\_POST['password2'])&&!empty($\_POST['indirizzo'])){          require 'application/libs/hash.php';          require 'application/libs/email.php';          require 'application/libs/password.php';          $nome = AntiCsScript::check($\_POST['nome']);          $cognome = AntiCsScript::check($\_POST['cognome']);          $email = AntiCsScript::check($\_POST['email']);          $pass1 = AntiCsScript::check($\_POST['password1']);          $pass2 = AntiCsScript::check($\_POST['password2']);          $indirizzo = AntiCsScript::check($\_POST['indirizzo']);          $emailUser = new Email($email);          if($emailUser->isValid()){              if($pass1 == $pass2){                  $passUser = new Password($pass1);                  if($passUser->isValid()){                      $hp = new Hash($pass1);                      $hp->doHash($email);                      $hash\_password = $hp->getHashed();                      $sql = $conn->prepare("INSERT INTO dipendente(nome, cognome, indirizzo, email, hash\_password, archiviato, datore\_id) VALUES (?, ?, ?, ?, ?, 0, ?)");                      $sql->bind\_param("sssssi",$nome, $cognome, $indirizzo, $email, $hash\_password, $\_SESSION['id']);                      $result = $sql->execute();                      return $email;                  }else{                      throw new Exception("La password deve contenere almeno:<br>- 8 Caratteri<br>-1 Maiuscola<br>-1 Minuscola<br>-1 Cifra<br>-1 Carattere speciale");                  }              }else{                  throw new Exception("Le due password inserite non corrispondono");              }          }      }else{          throw new Exception("Completare tutti i campi");      }  } |

La classe “dipendente\_model.php” viene utilizzata per effettuare le operazioni sul database che riguardano la gestione dei dipendenti.

Questo codice serve per aggiungere un nuovo dipendente. Anche qui i controlli sono quasi identici a quelli effettuati anche precedentemente per la modifica di un datore, l’unica differenza è che viene controllato direttamente che l’e-mail inserita sia unica e non sia già presente per qualche utente non archiviato.

**email\_model.php**

|  |
| --- |
| public function invia($destinatario, $nome) {      if (filter\_var($destinatario, FILTER\_VALIDATE\_EMAIL)) {          $oggetto = "Aggiornamento orario di lavoro";            $messaggio =  $this->content;          $messaggio = str\_replace("Salve!", "Salve $nome!", $messaggio);            $header = "From:gioele.zanetti@samtrevano.ch\r\n";          $header .= "MIME-Version: 1.0\r\n";          $header .= "Content-type: text/html\r\n";          echo $messaggio;            $retval = mail($destinatario,$oggetto,$messaggio,$header);            if( !$retval ) {              throw new Exception("Email con destinatario $destinatario non inviata" . PHP\_EOL);          }      }else{          throw new Exception("Email $destinatario non valida" . PHP\_EOL);      }  } |

Questa classe viene utilizzata per inviare un’e-mail di notifica al dipendente a cui è stato modificato il proprio orario di lavoro. Il metodo “invia” serve per appunto inviare al destinatario una pagina html fissa via e-mail.

**login\_model.php**

|  |
| --- |
| function doLogin(){      require 'application/libs/connection.php';      $this->getHashedPass();      $sql1 = $conn->prepare("SELECT \* FROM dipendente WHERE email=? AND hash\_password=? AND archiviato=0");      $sql1->bind\_param("ss", $this->email, $this->hashedPassword);      $sql2 = $conn->prepare("SELECT \* FROM datore WHERE email=? AND hash\_password=? AND archiviato=0");      $sql2->bind\_param("ss", $this->email, $this->hashedPassword);      $sql3 = $conn->prepare("SELECT \* FROM amministratore WHERE email=? AND hash\_password=?");      $sql3->bind\_param("ss", $this->email, $this->hashedPassword);      for($i=1;$i<=3;$i++){          $currentQuery = 'sql' . $i;          $$currentQuery->execute();          $result = $$currentQuery->get\_result();          if($result){              if ($result->num\_rows > 0) {                  $row = $result->fetch\_assoc();                  $\_SESSION['id'] = $row['id'];                  $\_SESSION['role'] = $i;                  return true;              }          }          $$currentQuery->close();      }      session\_destroy();      return false;  } |

Questo model contiene tutti i metodi per effettuare il login nell’applicativo web. Questa funzione legge in tutte e 3 le tabelle degli utenti (amministratore, datore e dipendente), e se trova che le credenziali corrispondono ad a qualcuno, permette il login salvano l’id e il ruolo dell’utente in una variabile di sessione.

**negozio\_model.php**

|  |
| --- |
| public function ottieniNegoziDipendente($id){      require 'application/libs/connection.php';      $query = "SELECT n.nome, n.id              FROM negozio n              INNER JOIN turno\_lavoro t              ON n.id = t.negozio\_id              WHERE n.archiviato = 0 AND t.dipendente\_id = $id              group by negozio\_id";      $result = $conn->query($query);      $data = array();      if ($result->num\_rows > 0) {          while($row = $result->fetch\_assoc()){              $data[] = $row;          }          return $data;      }      return false;  } |

La classe “negozio\_model” contiene tutte le funzioni per la gestione dei negozi.

Questo metodo serve per ottenere tutti i negozi di un certo dipendente tramite una query fatta al database, che contiene anche una join.

|  |
| --- |
| public function salva($input){      require 'application/libs/connection.php';      $orari = $this->ottieniOrari($input);      mysqli\_autocommit($conn, false);      $this->cominciaTransazione($conn);      $this->disattivaOrariNegozio($conn);      if($this->salvaOrari($conn, $orari)){          $this->commit($conn);          mysqli\_autocommit($conn, true);          $conn->query("call controlloDuplicati()");          $conn->close();          return "committed";      }else{          $this->rollback($conn);          mysqli\_autocommit($conn, true);          $conn->close();          return "rollback";      }  } |

Il metodo salva gli orari che l’utente vuole usare e ritorna uno status, che può essere negativo o positivo in base a com’è andato il salvataggio. Come prima cosa vengono presi gli orari da salvare, poi viene tolto l’autocommit, così in caso di errore si può effettuare un rollback. Se il salvataggio degli orari va a buon fine, viene eseguito il commit, altrimenti in caso di errore fa un rollback e ritorna lo status di rollback.

**orario\_model.php**

|  |
| --- |
| public function inserisciEventi($conn, $range, $eventi){      try{          $inizio = $this->formattaData($range['start']);          $fine = $this->formattaData($range['end']);          foreach($eventi as $evento){              if(strtotime($this->formattaData($evento['start'])) >= strtotime($inizio) &&              strtotime($this->formattaData($evento['start'])) >= strtotime($inizio) &&              strtotime($this->formattaData($evento['end'])) <= strtotime($fine) &&              strtotime($this->formattaData($evento['end'])) <= strtotime($fine)){                  $query = $conn->prepare("INSERT INTO turno\_lavoro VALUES (?, ?, ?, ?, ?)");                  $dipendente = $this->trovaId($evento['title']);                  $inizioEvento = $this->formattaOrario($evento['start']);                  $fineEvento =$this->formattaOrario($evento['end']);                  $data = $this->formattaData($evento['start']);                  $query->bind\_param("iisss", $dipendente, $\_SESSION['negozio\_id'], $inizioEvento, $fineEvento, $data);                  if(!$query->execute()){                      return false;                  }                  Log::writeLog("Nuovo evento inserito");              }          }          return true;      }catch(Exception $e){          return false;      }  } |

Questa classe contiene tutti i metodi necessari per manipolare i dati degli orari nel database.

La funzione qui sopra serve per inserire gli eventi all’interno del database. Viene passato un range nel quale i nuovi orari devono essere, e se ne fanno parte vengono aggiunti al database.

|  |
| --- |
| public function modificaOrario(){      if(!empty($\_POST['inizio']) && !empty($\_POST['fine'] && !empty($\_POST['orario']))){          $tinizio = mktime(substr($\_POST['inizio'], 0, 2), substr($\_POST['inizio'], 3, 2));          $tfine = mktime(substr($\_POST['fine'], 0, 2), substr($\_POST['fine'], 3, 2));          $inizio = $\_POST['inizio'] . ":00";          $fine = $\_POST['fine'] . ":00";          if($tinizio < $tfine){              $data = $this->ottieniOrari($inizio, $fine);              if($data == false){                  require 'application/libs/connection.php';                  $query = "UPDATE orario set inizio = '$inizio', fine = '$fine'                              WHERE id = ".$\_POST['orario'].";";                  $conn->query($query);                  $result = $conn->query($query);                  return $\_POST['orario'];              }else{                  throw new Exception("Questi orari sono già presenti");              }          }else{              throw new Exception("Inserire orari coerenti");          }      }else{          throw new Exception("Compilare tutti i campi");      }  } |

La funzione “modificaOrario” viene utilizzata, appunto, per cambiare un orario. Viene controllato che l’orario di inizio inserito sia minore di quello di fine, poi se l’orario che si cerca di modificare è presente nel database. Se tutto va a buon fine, vengono apportate le modifiche dell’orario nel database.

**pianifica\_model.php**

|  |
| --- |
| public function inserisciDipendente($id, $orario, $data, $negozio, $conn){      $query = $conn->prepare("INSERT INTO turno\_lavoro VALUES(?, ?, ?, ?, ?)");      $query->bind\_param("iisss", $id, $negozio, $orario['inizio'], $orario['fine'], $data);      $query->execute();      return $query->get\_result();  } |

La classe “pianifica\_model.php” serve per elaborare, inserire ed estrarre i dati dei dipendenti da inserire nella pianificazione. Nel metodo “inserisciDipendente” viene inserito il dipendente con l’orario di quando lavora, la data di quando lavora, il negozio di dove lavora e la connessione al DB.

|  |
| --- |
| public function inserisciDipendenti($negozio, $dipendenti, $inizio, $fine, $conn){      $data = $inizio;      $counterDipendenti = 0;      while(strtotime($data) <= strtotime($fine)){          $giorno = $this->daDataAGiorno($data);          $orari = $this->ottieniOrariDiNegozio($negozio, $giorno);          if($orari){              foreach($orari as $orario){                  $done = false;                  while(!$done){                      for($i=0;$i<2;$i++){                          $dipendente = $dipendenti[$counterDipendenti];                          $orariLavoroDipendente = $this->ottieniOrariLavoroDipendente(  $dipendente['id'],  $data);                          if(!$orariLavoroDipendente){                              $this->inserisciDipendente($dipendente['id'], $orario, $data,  $negozio, $conn);                              $done = true;                          }else{                              foreach($orariLavoroDipendente as $orarioLavoro){                                  if($this->dipendenteLibero($orario['inizio'], $orario['fine'],  $orarioLavoro['turno\_inizio'],  $orarioLavoro['turno\_fine'])){                                      $this->inserisciDipendente($dipendente['id'], $orario,  $data, $negozio, $conn);                                      $done = true;                                      break;                                  }                              }                              if($counterDipendenti == count($dipendenti) - 1){                                  throw new Exception("Tutti i dipendenti sono occupati durante l'orario " . $orario['inizio'] . "-" . $orario['fine'] . "!");                              }                          }                          $counterDipendenti = ($counterDipendenti + 1) % count($dipendenti);                      }                  }              }          }          $data = date('Y-m-d', strtotime($data. ' + 1 days'));      }  } |

Questa funzione usa la funzione “inserisciDipendente”. Questa funzione serve per riempire il calendario in modo automatico. Bisogna passare una data di inizio e fine, i dipendenti ed il negozio. In questo modo viene riempito il calendario entro le date specificate.

**tipi\_model.php**

|  |
| --- |
| public function aggiungi(){      require 'application/libs/connection.php';      if(!empty($\_POST['nome']) && !empty($\_POST['descrizione'])){          $nome = AntiCsScript::check($\_POST['nome']);          $descrizione = AntiCsScript::check($\_POST['descrizione']);          $sql = $conn->prepare("INSERT INTO tipo(nome, descrizione) VALUES (?, ?)");          $sql->bind\_param("ss",$nome, $descrizione);          $result = $sql->execute();          if($result){              return $nome;          }          throw new Exception("Errore durante l'inserimento del nuovo tipo");      }else{          throw new Exception("Completare tutti i campi");      }  } |

Questa classe serve a gestire i tipi dei negozi. In questa funzione, se tutti i controlli sulla stringa del nome e della descrizione vanno a buon fine, viene aggiunto un tipo al database.

**turno\_model.php**

|  |
| --- |
| public function turnoInRange($inizioTurno, $fineTurno){      require 'application/libs/connection.php';      $sql = $conn->prepare("SELECT o.inizio as inizio, o.fine as fine FROM usa      INNER JOIN orario o      ON o.id = usa.orario\_id      WHERE negozio\_id = ?");      $negozio = AntiCsScript::check($\_SESSION['negozio\_id']);      $sql->bind\_param("i", $negozio);      $sql->execute();      $result = $sql->get\_result();      if ($result->num\_rows > 0) {          while($row = $result->fetch\_assoc()){              $inizio = $this->daOrarioASecondi($row['inizio']);              $fine = $this->daOrarioASecondi($row['fine']);              if($inizio <= $this->daOrarioASecondi($inizioTurno) && $fine >= $this->daOrarioASecondi($fineTurno)){                  return true;              }          }      }      return false;  } |

“turno\_model” contiene tutte le funzioni per la gestione dei turni sul database. Questa funzione serve per trovare se un turno fa parte di uno specifico range oppure no.

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 | **Nome:** | Database funzionante |
| **Descrizione:** | Si controlla se è possibile il collegamento al database | | |
| **Prerequisiti:** | * Avere apache funzionante sulla porta 80 * Avere php funzionante * Avere mysql installato | | |
| **Procedura:** | 1. Visitare l’url gestionePersonale/db/test 2. Premere invio | | |
| **Risultati attesi:** | Si apre una pagina con scritto “Connected successfully” | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 | **Nome:** | Pagina di login funzionante |
| **Descrizione:** | Si controlla se il login viene effettuato correttamente per un amministratore | | |
| **Prerequisiti:** | * Avere il database funzionante * Avere un utente amministratore | | |
| **Procedura:** | 1. Aprire la pagina per il login 2. Inserire il nome utente e le credenziali dell’amministratore 3. Premere il tasto login | | |
| **Risultati attesi:** | Viene mostrata la pagina home dell’amministratore | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-003 | **Nome:** | Pagina home del datore di lavoro funzionante |
| **Descrizione:** | Si controlla se la pagina home del datore funziona correttamente | | |
| **Prerequisiti:** | * Avere il database funzionante * Avere un utente datore | | |
| **Procedura:** | 1. Aprire la pagina per il login 2. Inserire il nome utente e le credenziali dell’amministratore 3. Premere il tasto login | | |
| **Risultati attesi:** | Viene mostrata la pagina home del datore | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-004 | **Nome:** | Pagina home del dipendente funzionante |
| **Descrizione:** | Si controlla se la pagina home del dipendente funziona correttamente | | |
| **Prerequisiti:** | * Avere il database funzionante * Avere un utente datore | | |
| **Procedura:** | 1. Aprire la pagina per il login 2. Inserire il nome utente e le credenziali dell’amministratore 3. Premere il tasto login | | |
| **Risultati attesi:** | Viene mostrata la pagina home del dipendente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-005 | **Nome:** | Invio della mail ai dipendenti |
| **Descrizione:** | Si controlla se viene inviata una mail quando un datore modifica l’orario di un dipendente | | |
| **Prerequisiti:** | * Avere dei dipendenti da spostare * Fare login con un account datore | | |
| **Procedura:** | 1. Selezionare il negozio di cui si vuole modificare l’orario 2. Spostare alcuni dipendenti in altri orari 3. Salvare le modifiche | | |
| **Risultati attesi:** | Nell’inbox della mail del dipendente c’è una mail che notifica del cambio di orario | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-006 | **Nome:** | Controllo funzionamento delle statistiche |
| **Descrizione:** | Si controlla se le statistiche funzionano in modo corretto | | |
| **Prerequisiti:** | * Avere dei dipendenti * I dipendenti hanno lavorato per il datore | | |
| **Procedura:** | 1. Navigare fino alla pagina di statistiche 2. Inserire una data di inizio per la statistica 3. Inserire una data di fine per la statistica 4. Fare clic sul bottone per visualizzare i risultati | | |
| **Risultati attesi:** | Nella pagina vengono visualizzate le statistiche per ogni dipendente: ore di lavoro durante i giorni feriali, festivi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-007 | **Nome:** | Controllo del funzionamento dell’algoritmo di popolazione |
| **Descrizione:** | Si controlla che l’algoritmo funzioni in modo corretto | | |
| **Prerequisiti:** | * Avere tre dipendenti per lo stesso negozio * Fare login come datore | | |
| **Procedura:** | 1. Nella pagina home, inserire il periodo di una settimana 2. Premere sul bottone pianifica | | |
| **Risultati attesi:** | L’orario settimanale del negozio viene popolato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-008 | **Nome:** | Creazione nuovo negozio |
| **Descrizione:** | Si controlla che vengano creati negozi correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore | | |
| **Procedura:** | 1. Navigare fino alla pagina per aggiungere un negozio. 2. Inserire il nome “Panetteria Donofrio” 3. Inserire vari dati 4. Premere sul bottone “Aggiungi” | | |
| **Risultati attesi:** | Nella lista dei negozi posseduti viene aggiunto il negozio “Panetteria Donofrio” | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-008 | **Nome:** | Modifica negozio |
| **Descrizione:** | Si controlla che un i dati di negozio vengano modificati correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere “Panetteria Donofrio” nel database | | |
| **Procedura:** | 1. Navigare fino alla pagina per modificare un negozio. 2. Cambiare il nome del negozio “Panetteria Donofrio” a “Panificio Donofrio” 3. Premere sul bottone “Modifica” | | |
| **Risultati attesi:** | Nella lista dei negozi posseduti viene modificato il negozio “Panetteria Donofrio” in “Panificio Donofrio”. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-008 | **Nome:** | Eliminazione di un negozio |
| **Descrizione:** | Si controlla che i negozi vengano eliminati correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere almeno un negozio da eliminare | | |
| **Procedura:** | 1. Navigare fino alla pagina per eliminare un negozio. 2. Selezionare il negozio da eliminare 3. Premere sul bottone elimina | | |
| **Risultati attesi:** | Nella lista dei negozi posseduti non si vede più il negozio appena eliminato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-009 | **Nome:** | Aggiunta dipendenti |
| **Descrizione:** | Si controlla che i dipendenti vengano aggiunti correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore | | |
| **Procedura:** | 1. Navigare fino alla pagina per aggiungere un dipendente. 2. Inserire il nome “Diego Donofrio” 3. Inserire i dati rimanenti | | |
| **Risultati attesi:** | Nella lista dei dipendenti disponibili viene aggiunto Diego Donofrio | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-012  REQ-009 | **Nome:** | Modifica dipendente |
| **Descrizione:** | Si controlla che i dati di un dipendente vengano modificati correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere “Diego Donofrio” nel database | | |
| **Procedura:** | 1. Navigare fino alla pagina per modificare un dipendente. 2. Cambiare il nome del dipendente da “Diego Donofrio” in “Daniele Donofrio” 3. Premere sul bottone “Modifica” | | |
| **Risultati attesi:** | Nella lista dei dipendenti disponibili viene modificato il negozio “Diego Donofrio” in “Daniele Donofrio”. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-013  REQ-009 | **Nome:** | Eliminazione di un dipendente |
| **Descrizione:** | Si controlla che i dipendenti vengano eliminati correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere almeno un dipendente da eliminare | | |
| **Procedura:** | 1. Navigare fino alla pagina per eliminare un dipendente. 2. Selezionare il dipendente da eliminare 3. Premere sul bottone elimina | | |
| **Risultati attesi:** | Nella lista dei dipendenti disponibili non si vede più il dipendente appena eliminato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-014  REQ-010 | **Nome:** | Creazione orario di apertura |
| **Descrizione:** | Si controlla che un orario di apertura venga creato correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore | | |
| **Procedura:** | 1. Navigare fino alla pagina per aggiungere un orario. 2. Inserire 07:00 e 12:30 3. Premere sul bottone aggiungi | | |
| **Risultati attesi:** | Nella lista degli orari compare quello che abbiamo inserito | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-015  REQ-010 | **Nome:** | Modifica orario di apertura |
| **Descrizione:** | Si controlla che un orario di apertura venga modificato correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere l’orario 07:00-12:30 nel database | | |
| **Procedura:** | 1. Navigare fino alla pagina per modificare un orario. 2. Selezionare l’orario dalle 08:00 alle 12:30 3. Modificare in 09:00-12:30 4. Premere sul bottone Modifica | | |
| **Risultati attesi:** | Nella lista degli orari compare quello che abbiamo modificato e non più quello precedente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-016  REQ-010 | **Nome:** | Aggiunta Turno |
| **Descrizione:** | Si controlla che un turno venga creato correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere nel database Panetteria Donofrio * Avere un turno di apertura il martedì almeno dalle 08:00 alle 11:00 | | |
| **Procedura:** | 1. Navigare fino alla pagina per aggiungere un turno. 2. Selezionare martedi 3. Inserire un turno dalle 08:00 alle 11:00 | | |
| **Risultati attesi:** | Nella lista dei turni compare il turno appena inserito | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-017  REQ-010 | **Nome:** | Eliminazione Turno |
| **Descrizione:** | Si controlla che un turno venga eliminato correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere nel database Panetteria Donofrio * Avere un turno 08:00-11:00 | | |
| **Procedura:** | 1. Navigare fino alla pagina per eliminare un turno. 2. Eliminare il turno dalle 08:00-11:00 | | |
| **Risultati attesi:** | Nella lista non compare più il turno | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-018  REQ-011 | **Nome:** | Visualizzazione calendario datore |
| **Descrizione:** | Si controlla che il calendario del datore funzioni correttamente | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere Panetteria Donofrio nel database * Avere un dipendente associato a Panetteria Donofrio * Avere un turno di lunedì dalle 08:00 alle 10:00 | | |
| **Procedura:** | 1. Andare nella pagina per visualizzare il calendario 2. Scegliere un negozio 3. Inserire un dipendente lunedì dalle 08:00 alle 10:00 4. Salvare 5. Ricaricare la pagina | | |
| **Risultati attesi:** | Il turno compare ancora dopo aver ricaricato la pagina | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-019  REQ-011 | **Nome:** | Visualizzazione calendario dipendente |
| **Descrizione:** | Si controlla che il calendario del dipendente funzioni correttamente | | |
| **Prerequisiti:** | * Fare login con un account dipendente * L’account dipendente deve essere associato ad un negozio * Il dipendente deve avere un turno impostato dal datore | | |
| **Procedura:** | 1. Scegliere un negozio 2. Navigare finché non si vedono i propri turni | | |
| **Risultati attesi:** | Il dipendente vede correttamente il suo turno | | |

## Risultati test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case** | **Esito** | **Risultati** | **Data** |
| TC-001 | **PASSATO** | La pagina mostra che il database è connesso | 28.04.2022 |
| TC-002 | **NON PASSATO** | Dopo aver effettuato il login come amministratore, viene sollevata questa eccezione | 28.04.2022 |
| TC-003 | **PASSATO** | La pagina home del datore viene visualizzata correttamente | 28.04.2022 |
| TC-004 | **PASSATO** | La pagina home del dipendente viene visualizzata correttamente | 28.04.2022 |
| TC-005 | **PASSATO** | L’email viene inviata | 28.04.2022 |
| TC-006 | **NON PASSATO** | La pagina non si apre proprio. Viene effettuato un redirect verso la pagina home | 28.04.2022 |
| TC-007 | **PASSATO** | L’intervallo i tempo specificato viene pianificato in modo corretto | 28.04.2022 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-008 | **PASSATO** | Il negozio viene aggiunto correttamente | 28.04.2022 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-009 | **PASSATO** | Il negozio viene modificato correttamente | 28.04.2022 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-010 | **PASSATO** | Il negozio viene eliminato correttamente | 28.04.2022 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TC-011 | **PASSATO** | Il dipendente viene creato correttamente | 28.04.2022 | |
| TC-012 | **PASSATO** | Il dipendente viene modificato correttamente | | 28.04.2022 | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-013 | **PASSATO** |  | 28.04.2022 |
| TC-014 | **NON PASSATO** | Vengono creati dei doppioni quando viene inserito un orario | 28.04.2022 |
| TC-014 | **PASSATO** | L’orario viene aggiunto correttamente | 05.05.2022 |
| TC-015 | **PASSATO** | L’orario viene modificato | 28.04.2022 |
| TC-016 | **PASSATO** | Il turno viene inserito correttamente | 28.04.2022 |
| TC-17 | **NON PASSATO** | Il turno non viene eliminato e non si sa il perché, non viene prodotto nessun errore | 05.05.2022 |
| TC-18 | **PASSATO** | I turni dei dipendenti vengono inseriti correttamente | 05.05.2022 |
| TC-19 | **PASSATO** | Il dipendente vede i propri turni | 05.05.2022 |

## Mancanze/limitazioni conosciute

Fortunatamente, siamo riusciti ad implementare quasi tutto quello che ci è stato richiesto nel mandato. L’unica cosa che non abbiamo fatto è stata la possibilità di impostare le ore di lavoro settimanali e i giorni festivi dell’anno.

Ci sono alcune mancanze dovute alla scarsa conoscenza del linguaggio PHP da parte di tutti, ma grazie a questo progetto siamo riusciti a migliorarle.

# Consuntivo

# Conclusioni

## Sviluppi futuri

Sicuramente sarebbe quello di implementare la possibilità di inserire le ore di lavoro settimanali di un negozio e i giorni festivi del Canton Ticino, dato che è l’unico punto delle specifiche che non siamo riusciti ad implementare.

Si potrebbe aggiungere che ad ogni nuova registrazione di utente viene inviata un’email di notifica all’utente appena creato. Inserire anche le statistiche per i vari negozi, e non soltanto sui dipendenti.

Molto utile sarebbe anche aggiornare la grafica rendendola più semplice e pulita.

## Considerazioni personali

**Andrea**:

Per concludere posso dire che il progetto è servito per migliorare a lavora in modo più organizzato e ordinato, dovendo rispettare dei tempi di consegna e le varie specifiche date dal cliente.

Ho migliorato la mia conoscenza di PHP e MySQL, utilizzati per collegare il database al sito web.

Non ho avuto particolari problemi soprattutto perché all’inizio di questo progetto avevo appena terminato un progetto simile, sempre svolto con PHP.

Il risultato finale del progetto mi soddisfa, soprattutto perché inizialmente pensavo fosse molto più complicato, ma grazie alla suddivisione dei compiti siamo riusciti a completarlo senza troppi problemi. L’unico cosa che mi spiace è non essere riusciti ad implementare la possibilità di impostare le ore settimanali di lavoro di un negozio e i giorni festivi.

**Gioele:**

Secondo me, abbiamo gestito bene questo progetto. Io e i miei compagni siamo riusciti bene a suddividerci i compiti e a lavorare insieme, anche se all’inizio dovevamo ancora abituarci a lavorare in team. Questo progetto mi è stato utile per migliorare le mie conoscenze di PHP soprattutto, visto che non avevo mai approfondito questo linguaggio. Posso dire che sono soddisfatto, abbiamo raggiunto quasi tutti gli obbiettivi prefissati nonostante la difficoltà del progetto.

# Glossario

|  |  |
| --- | --- |
| **Nome** | **Definizione** |
| **Abstract** | Breve riassunto che spiega il progetto |
| **Applicativo WEB** | Il sito web che verrà visualizzato a progetto terminato |
| **Cliente** | La persona o azienda che finanzia il progetto e che detta le specifiche dei requisiti |
| **Controller** | È una classe che collega la parte grafica con la parte di modifica dei dati di un database. |
| **Database** | Struttura preparata su tabelle che permette lo storage di informazioni riguardanti un applicativo o progetto e rende possibile il reperimento dei dati voluti. |
| **Date** | Tipo di dato che salva unicamente la data in SQL |
| **Gantt** | Diagramma per pianificare un progetto dividendolo in varie attività, per avere un lavoro più organizzato |
| **Interfaccia** | L'hardware e il software che permettono a un utente di interagire con il computer. |
| **Log** | File che contiene tutte le informazioni per l’analisi e il troubleshooting |
| **Model** | Le classi che interagiscono col database |
| **MVC** | Pattern utilizzato per dividere le funzionalità ben distinte di un applicativo |
| **Pagina home** | Pagina iniziale di un sito Web, nella quale sono contenuti i collegamenti alle altre pagine del sito |
| **PHP** | Linguaggio di programmazione lato Server per la gestione degli applicativi  WEB |
| **Sito internet** | Sistema informatico che mette a disposizione informazioni e servizi mediante la rete Internet |
| **Use case** | È un metodo per raffigurare i requisiti raccolti, mostrando le possibili interazioni tra gli utenti ed il sistema |
| **View** | Pagine html che comunicano con i Controller |
| **MySQL** | Sistema open source per la gestione dei database |
| **Specifiche dei requisiti** | Descrizione completa del comportamento di un sistema software da sviluppare |

# Sitografia

* <https://httpd.apache.org/>, 10.02.2022
* <https://stackoverflow.com/>, 28.04.2022
* <https://www.php.net/>, 21.04.2022
* <https://www.w3schools.com/>, 21.04.2022
* <https://www.html.it/pag/18299/il-pattern-mcv/>, *Pattern MVC*, 31.03.2021
* <https://fullcalendar.io/>, 28.04.2022
* <https://it.wikipedia.org/>, 05.05.2022

# Allegati

Allegati:

* Diari di lavoro
* Codici sorgente presente su GitHub
* Quaderno dei compiti
* Manuale di sistema
* Gantt