Gestione Turni Personale

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

Analisi 4

1.4 Analisi del dominio 4

1.5 Analisi e specifica dei requisiti 4

1.6 Use case 6

1.7 Pianificazione 6

1.8 Analisi dei mezzi 6

1.8.1 Software 6

1.8.2 Hardware 6

2 Progettazione 7

2.1 Design dell’architettura del sistema 7

2.2 Design dei dati e database 7

2.3 Design delle interfacce 7

2.4 Design procedurale 7

3 Implementazione 8

4 Test 8

4.1 Protocollo di test 8

4.2 Risultati test 9

4.3 Mancanze/limitazioni conosciute 9

5 Consuntivo 9

6 Conclusioni 9

6.1 Sviluppi futuri 9

6.2 Considerazioni personali 9

7 Bibliografia 9

7.1 Bibliografia per articoli di riviste: 9

7.2 Bibliografia per libri 9

7.3 Sitografia 10

8 Allegati 10

# 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. |

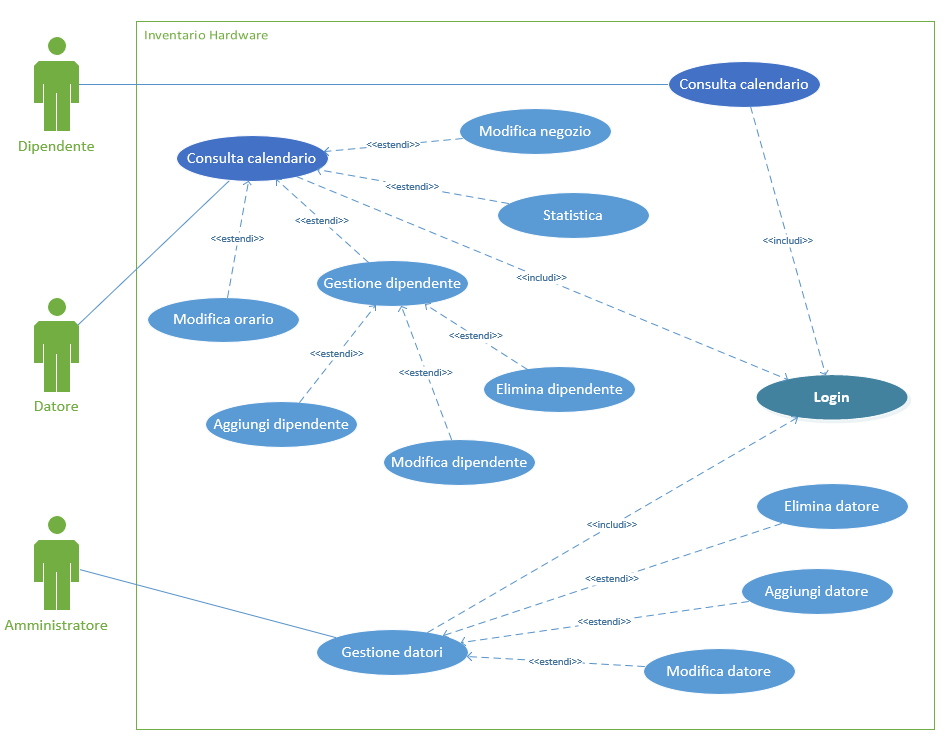
|  |  |
| --- | --- |
| ID: REQ-007 | |
| Nome | Algoritmo di popolazione dei negozi. |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita un algoritmo per organizzare in modo automatico gli orari dei dipendenti. |
| Sotto requisiti | |
| 001 | Conoscenza di PHP, HTML e MySQL. |

|  |  |
| --- | --- |
| 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 | Pagina panoramica dipendenti |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si necessita una pagina per la panoramica dei dipendenti. |
| Sotto requisiti | |
| 001 | Si necessita una maschera di login. |
| 002 | Si necessitano due campi che definiscono la data d’inizio e fine della statistica da visualizzare. |

## Use case



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

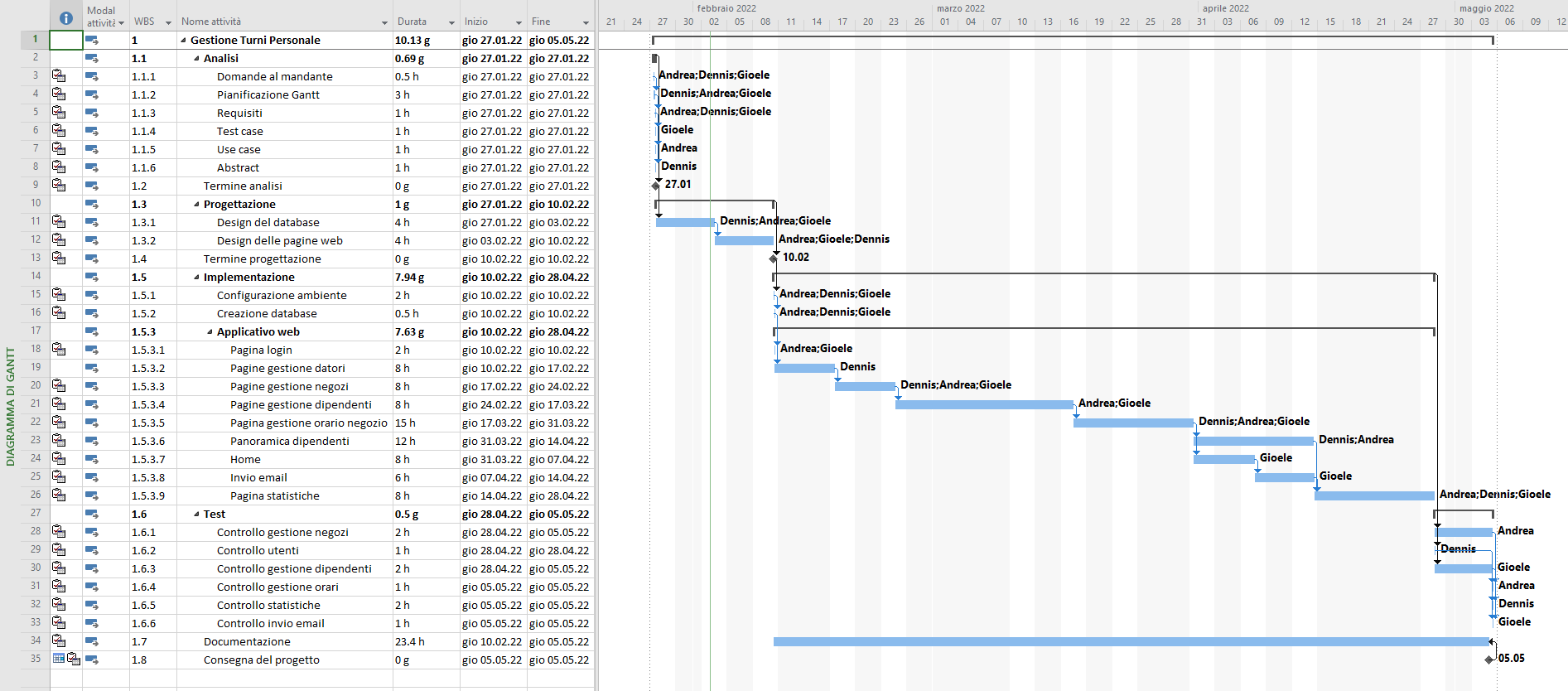
* **Dipendente**: L’unica interazione che questo utente avrà è consultare il calendario per vedere i propri turni di lavoro con i rispettivi orari.
* **Datore**: Questo utente, invece, ha il controllo completo sui propri dipendenti e negozi, infatti può aggiungere, eliminare o modificare sia gli impiegati 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

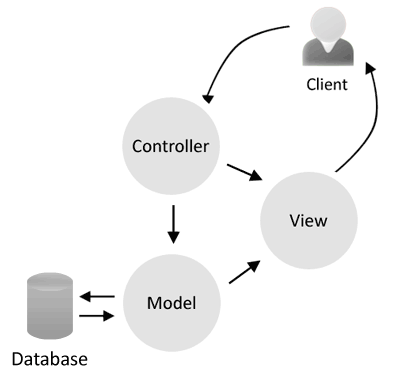
# Progettazione

## 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

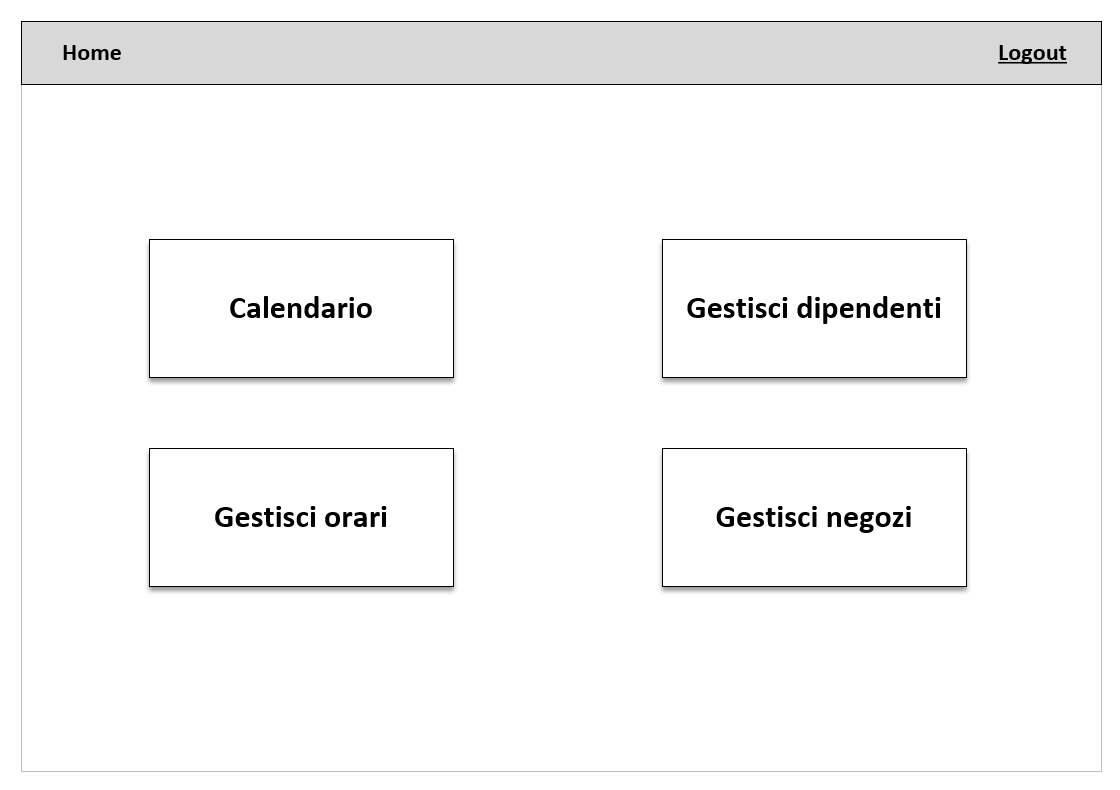
La struttura del database è piuttosto complessa, ed è formata da 11 tabelle, necessarie per il buon salvataggio delle informazioni. Le tabelle sono le seguenti:

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.

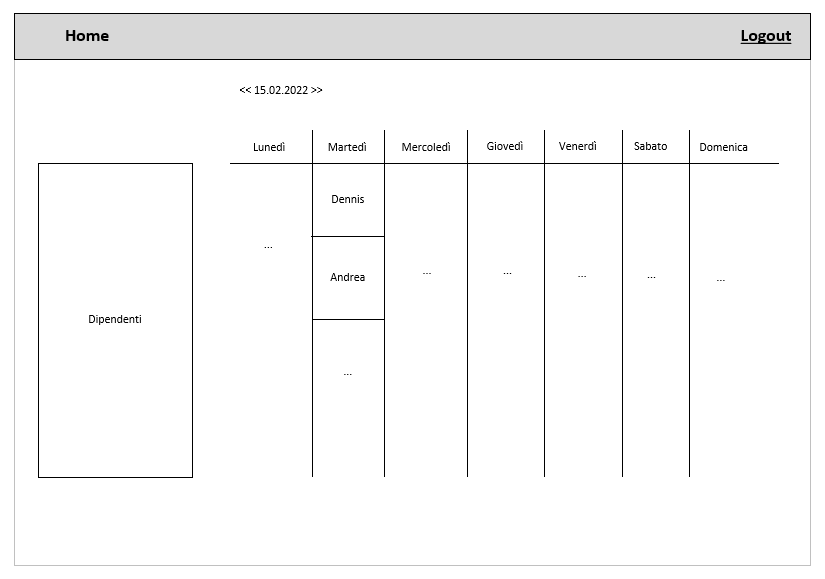
## Design delle interfacce

**Login**

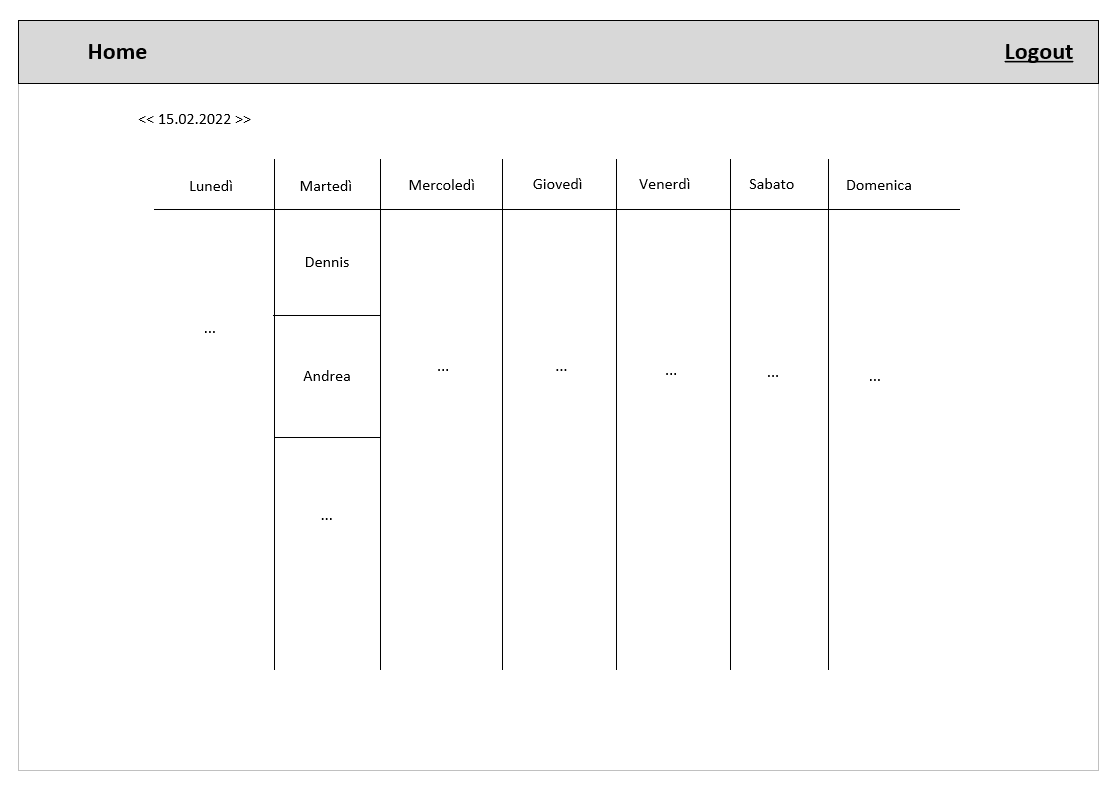
La pagina di login è classica, dove vengono chieste le credenziali di accesso, ed in base al tipo di utente che effettua l’accesso (Amministratore, datore o dipendete), viene aperta una home diversa.

**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.

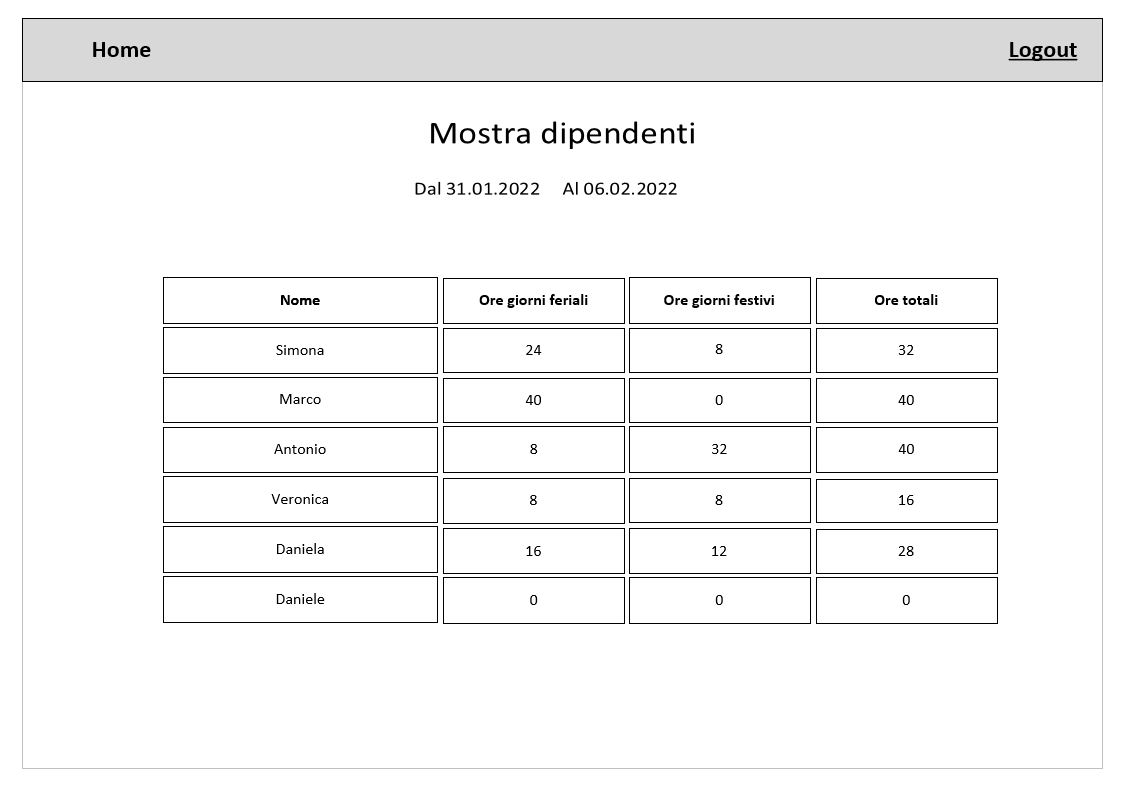
**Modifica turni di lavoro**

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**



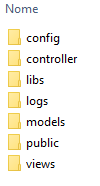
Si possono consultare tutti gli orari disponibili da impostare per i vari negozi.

# 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).

# 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. Creare un file .php e scrivere dentro il seguente codice:   **<?php $servername = "localhost"; $username = <username>; $password = <password>;  $conn = new mysqli($servername, $username, $password);  if ($conn->connect\_error) {   die("Connection failed: " . $conn->connect\_error); } echo "Connected successfully"; ?>**   1. Mettere il file nella cartella dei file web di apache 2. Aprire il browser 3. Cercare “localhost” | | |
| **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 | | |
| **Prerequisiti:** | * Avere il database funzionante * Avere un utente nel database | | |
| **Procedura:** | Aprire la pagina per il login  Inserire il nome utente e le credenziali dell’utente  Premere il tasto login | | |
| **Risultati attesi:** | Viene mostrata la pagina home | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **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:** | * Fare login con un account datore | | |
| **Procedura:** | Selezionare il negozio di cui si vuole modificare l’orario  Spostare alcuni dipendenti in altri orari  Salvare le modifiche | | |
| **Risultati attesi:** | Le modifiche vengono salvate correttamente e ogni volta che si visita la pagina le modifiche vengono visualizzate correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **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:** | * Fare login con un account dipendente | | |
| **Procedura:** | Navigare fino alla pagina per vedere il proprio orario | | |
| **Risultati attesi:** | L’orario del dipendente viene visualizzato correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **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:** | Selezionare il negozio di cui si vuole modificare l’orario  Spostare alcuni dipendenti in altri orari  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:** | Navigare fino alla pagina di statistiche  Inserire una data di inizio per la statistica  Inserire una data di fine per la statistica  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 e di lavoro | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **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:** | Nella pagina home, inserire il periodo di una settimana  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:** | Navigare fino alla pagina per aggiungere un negozio.  Inserire il nome “Panetteria Donofrio”  Inserire vari dati  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:** | Navigare fino alla pagina per modificare un negozio.  Cambiare il nome del negozio “Panetteria Donofrio” a “Panificio Donofrio”  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:** | Navigare fino alla pagina per eliminare un negozio.  Selezionare il negozio da eliminare  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:** | Navigare fino alla pagina per aggiungere un dipendente.  Inserire il nome “Diego Donofrio”  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:** | Navigare fino alla pagina per modificare un dipendente.  Cambiare il nome del dipendente da “Diego Donofrio” in “Dennis Donofrio”  Premere sul bottone “Modifica” | | |
| **Risultati attesi:** | Nella lista dei dipendenti disponibili viene modificato il negozio “Diego Donofrio” in “Dennis 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:** | Navigare fino alla pagina per eliminare un dipendente.  Selezionare il dipendente da eliminare  Premere sul bottone elimina | | |
| **Risultati attesi:** | Nella lista dei dipendenti disponibili non si vede più il dipendente appena eliminato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-013  REQ-009 | **Nome:** | Panoramica dipendenti funzionante |
| **Descrizione:** | Si controlla che le statistiche sulle ore di lavoro dei dipendenti vegano fatte in modo corretto | | |
| **Prerequisiti:** | * Fare login con un account datore * Avere almeno un negozio attivo * Avere dei dipendenti che lavorano nel negozio | | |
| **Procedura:** | Navigare fino alla pagina per fare statistiche.  Inserire una data di inizio e una data di fine  Premere sul bottone “Esegui statistica” | | |
| **Risultati attesi:** | Viene visualizzata una pagina in cui le ore in cui i dipendenti hanno lavorato vengono suddivise in ore in giorni lavorativi, ore in giorni festivi e ore in giorni feriali | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

# Glossario

|  |  |
| --- | --- |
| **Nome** | **Definizione** |
| **Applicativo WEB** | Il sito web che verrà visualizzato a progetto terminato |
| **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. |
| **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 |
| **Log** | File che contiene tutte le informazioni per l’analisi e il troubleshooting |
| **MVC** | Pattern utilizzato per dividere le funzionalità ben distinte di un applicativo |

# 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

# Allegati

Allegati:

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