Diario di lavoro

Luogo	Balerna
Data	18.05.2020

Lavori svolti

Durante la giornata di oggi mi sono occupato di aggiornare ed aggiungere i seguenti capitoli della documentazione:

- Validazione dei dati
- Gestione dati ed interrogazione database
- Sicurezza
    - Interrogazione database
    - Salvataggio dei dati
    - Salvataggio credenziali
    - Recupero password

Inoltre, come stabilito dalla pianifica mi sono occupato di terminare la gestione degli utenti ed inoltre ho terminato anche la gestione delle impostazioni dell'applicativo. Per implementare la gestione degli utenti avendo già la classe Model di gestione mi sono occupato di sviluppare il controller contenente i vari metodi di aggiornamento, inserimento e modifica. Ho dunque creato il controller User il quale contiene i metodi: insert, update e delete. Questi metodi utilizzano la classe Users per interagire con il database, ad esempio l'inserimento di un utente è il seguente:

```php
public static function insert(Request $req, Response $res) {
    $name = $req->getParam('name');
    $lastname = $req->getParam('lastname');
    $email = $req->getParam('email');
    try {
        if (
            Validator::isValidName($name)
            && Validator::isValidLastName($lastname)
            && Validator::isValidEmail($email)
            && Users::insert($email, $name, $lastname)
        ) {
            return $res->withStatus(200);
        }
    } catch (Exception $e) {
        return $res->withStatus(500)->withText($e->getMessage());
    }
    return $res->withStatus(400);
}
```

Il metodo ricava i parametri che vengono passati con la richiesta, controlla la validità dei dati e prova ad inserire l'utente. In caso di successo la pagina ritornerà il codice 200, in caso di errore il codice 500 mentre se mancano alcuni dati il codice 400.

Successivamente ho creato la classe Model per la gestione delle sezioni, questa classe è molto simile alle altre classi Model. Contiene metodi che permettono di gestire i dati delle sezioni, come ad esempio il metodo getAll che permette di ricavare tutte le sezioni presenti:

```php
public static function getAll()
{
    $pdo = Database::getConnection();
    $query = "SELECT * FROM section";
    try {
        $stm = $pdo->query($query);
        return $stm->fetchAll(\PDO::FETCH_ASSOC);
    } catch (\PDOException $e) {
        return false;
    }
}
```

Come per la gestione utenti ho creato una classe controller per le sezioni. In questo controller sono presenti due metodi: insert e delete. Il metodo insert è molto simile a quello degli utenti:

```php
public static function insert(Request $req, Response $res)
{
    $name = $req->getParam('name');
    try {
        if (Validator::isValidSection($name) && Sections::insert($name)) {
            return $res->withStatus(200);
        }
    } catch (Exception $e) {
        return $res->withStatus(500)->withText($e->getMessage());
    }
    return $res->withStatus(400);
}
```

Il metodo ricava i parametri che vengono passati con la richiesta, controlla la validità dei dati e prova ad inserire la nuova sezione. In caso di successo la pagina ritornerà il codice 200, in caso di errore il codice 500 mentre se mancano alcuni dati il codice 400. Come per la gestione delle sezioni e degli utenti, ho creato una classe Model per la gestione delle impostazioni. La classe Settings permette di: ricavare tutte le impostazioni, ricavare il valore di una impostazione e modificarne il valore. Per eseguire la modifica di un' impostazione il codice è il seguente:

```php
public static function update($name, $value)
{
    $setting = self::get($name);

    if ($setting) {
        if (
            call_user_func_array(
                array("FilippoFinke\Libs\Validator", "isValid" . $setting["type"]),
                array($value)
            )
        ) {
```

```
            $pdo = Database::getConnection();
            $query = "UPDATE setting SET value = :value WHERE name = :name";
            try {
                $stm = $pdo->prepare($query);
                $stm->bindParam(':value', $value);
                $stm->bindParam(':name', $name);
                return $stm->execute();
            } catch (\PDOException $e) {
                return false;
            }
        } else {
            throw new Exception("Il valore deve essere di tipo: " . $setting["type"]);
        }
    } else {
        throw new Exception("Impostazione inesistente");
    }
}
```

Quando viene eseguito l'aggiornamento di un'impostazione viene prima controllata la validità del nuovo valore chiamando in modo dinamico la classe validator passandone il tipo. Anche per questa tabella ho dunque creato una classe controller chiamata Setting la quale contiene solamente un metodo che permette di aggiornare il valore di un'impostazione:

```
public static function update(Request $req, Response $res)
{
    $setting = $req->getAttribute('setting');
    $value = $req->getParam('value');
    try {
        if ($value !== null && Settings::update($setting, $value)) {
            return $res->withStatus(200);
        }
    } catch (Exception $e) {
        return $res->withStatus(500)->withText($e->getMessage());

    }
    return $res->withStatus(400);
}
```

Anche in questo caso i codici di stato sono gli stessi a quelli delle altre classi controller. In fine ho implementato la classe Model ed il Controller per la tabella year la quale permetterà di ricavare i semestri. Anche in questo caso i metodi sono simili e sullo stesso concetto delle altre classi.

Problemi riscontrati e soluzioni adottate

Non ho riscontrato nessun problema nello sviluppo della parte di gestione di utenti ed impostazioni.

Punto della situazione rispetto alla pianificazione

Avendo terminato la gestione degli utenti e la gestione delle impostazioni al momento ho circa 2 ore di vantaggio rispetto alla pianifica.

Programma di massima per la prossima giornata di lavoro
Iniziare lo sviluppo del sistema di gestione dei ritardi.