Diario di lavoro

Luogo	Canobbio, SAM Trevano
Data	24/10/2019

Lavori svolti

All'interno della classe **Model** "Validator" sono presenti diversi tipi di validazione sui dati o altri metodi che aiutano la validazione.

Questo metodo permette di filtrare in generale qualsiasi dato, togliendo qualsiasi carattere illegale che potrebbe comportare un inaspettato comportamento del sito.

```
/**
 * Validate data of any kind.
*
* @param data The data to validate
* @return boolean The validated data
*/
private function generalValidation($data)
{
    $data = trim(stripslashes(htmlspecialchars($data)));
    return $data;
}
```

Ho cercato una soluzione su internet, di come sapere se una email è valida o non.
Link: https://stackoverflow.com/questions/5855811/how-to-validate-an-email-in-php

Questo metodo permette di sapere, se una email è valida strutturalmente e sintatticamente.
Se ritorna "**TRUE**" allora significa che è valida, ma deve essere comunque filtrata in ogni caso.

```
/**
 *  Return true if the given email is valid, structurally and syntactically.
 *
 * @param email The email to check the validation
 * @return boolean If the email is valid return true otherwise false
 */
function isValidEmail($email)
{
    return filter_var($email, FILTER_VALIDATE_EMAIL) &&
                preg_match('/@.+\./', $email);
}
```

Questo metodo filtra i caratteri dell'email e se la filtrazione fallisce ritorna "**FALSE**", invece se ci riesce ritorna il testo filtrato.

```
filter_var($email, FILTER_VALIDATE_EMAIL)
```

Questa parte invece verifica la struttura dell'email, verificando che i caratteri obbligatori ci siano e nella posizione giusta all'interno dell'email.

```
preg_match('/@.+\./', $email);
```

Questo metodo ritorna l'email validata, grazie al metodo "filter_var(<variabile>,<tipoFiltro>)" che filtra ogni singolo carattere dell'email, e se la validazione fallisce ritorna "**FALSE"**.

```
/**
 * Validate data of type string.
 *
 * @param email The email to validate
 * @return boolean The validated email
 */
public function validateEmail($email)
{
    $validEmail = $this->generalValidation($email);
    return filter_var($validEmail, FILTER_VALIDATE_EMAIL);
}
```

Questo metodo permette di validare un numero intero, inserendo un qualsiasi numero ritorna un intero validato.

```
/**
 * Validate data of type int.
 *
 * @param i The integer number to validate
 * @return boolean The validated data
 */
public function validateInt($i)
{
    $validElement=$this->generalValidation($element);
    return intval($validElement);
}
```

Questo metodo permette di validare una stringa, inserendo un testo qualsiasi ritorna una stringa filtrata.

```
/**
 * Validate data of type string.
 *
 * @param str The string to validate
 * @return boolean The validated string
 */
public function validateString($str)
{
    $validStr = $this->generalValidation($str);

    $pattern = '/^[A-Za-z0-9_-.àèìòùÀÈÌÒÙáéíóúýÁÉÍÓÚÝâêîôûÂÊÎÔÛãñõÃÑÕäëïöüÿÄËÏÖÜŸçÇßØøÅåÆæœ]*$/';

    if (!preg_match($pattern, $validStr))
    {
        $validStr = strval($validStr);
    }
    return $validStr;
}
```

Ho fatto questa *query* sul database, per inserire le credenziali dell'amministratore nel DB.

```
insert into administrator values(null,'Administrator',sha2('Invoices2019',256),'administrator@gmail.com');
```


Problemi riscontrati e soluzioni adottate
Ho cambiato il tipo di dato in cui è memorizzata una password nel database, sia per la tabella “User” che della tabella “Administrator”, ora si tratta di un char(64). Questo cambiamento è stato necessario perché il tipo binary(32) è troppo piccolo, per memorizzare una hash codificata in sha256.

Punto della situazione rispetto alla pianificazione
In linea con la pianificazione.

Programma di massima per la prossima giornata di lavoro
Continuare l’implementazione, completare l’interfaccia di registrazione e login; documentare il codice