

LEAVE FORM MANAGEMENT

A MINI PROJECT REPORT

Submitted by

K J JAMES (10MSE1025)

HARSHAVARDHAN REDDY G(10MSE1077)

in partial fulfillment for the award of degree of

M.S Software Engineering



School of Computing Science and Engineering

VIT Chennai, Chennai-600127, Tamil Nadu, India

May, 2014



School of Computing Science and Engineering

DECLARATION

We hereby declare that the project entitled **“Leave Form Management”** submitted by me/us to the School of Computing Science and Engineering, VIT University, Chennai Campus, Chennai – 127 in partial fulfillment of the requirements for the award of the degree of **M.S Software Engineering** is a record of bonafide work carried out by me/us under the supervision of Sivagami M, **Designation**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

HARSHAVARDHAN REDDY G(10MSE1077)

K J JAMES (10MSE1025)



School of Computing Science and Engineering

CERTIFICATE

The project report entitled “**Leave Form Management**” is prepared and submitted by **Candidate HARSHAVARDHAN REDDY G(10MSE1077) and K J JAMES (10MSE1025)**. It has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirements for the award of the degree of **M.S Software Engineering** in VIT University, Chennai Campus, Chennai, India.

Signature
(Name of the Supervisor/Guide)

Examined by:

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere gratitude to Prof. Sivagami M for her invaluable mentoring and immense patience.

I would also like to thank Dr.N.Maheswari for her support and guidance throughout the project.

I would like to thank Dr.L Jegannathan for his support and guidance throughout the project.

CONTENTS

Chapter	Title	Page
	Title Page	i
	Declaration	Ii
	Certificate	Iii
	Acknowledgement	Iv
	Table of contents	V
	List of Tables	Vi
	List of Figures	Vi
	List of Abbreviation	Vii
	Abstract	viii
1	Introduction	1
2	Analysis and Design	6
	2.1 Introduction to problem definition	6
	2.2 Proposed System Architecture	6
	2.3 Requirement Analysis	9
	2.4 Detailed Design	10
3	Implementation and Testing	11
	3.1 Modules	11
	3.2 Technical Modules	12
	3.3 Unit test cases	20
4	Results and Discussion	21
5	Conclusion and Future Enhancements	24
	References	25

LIST OF TABLES

	Title	Page
	Table 1.1 : Unit test cases	

LIST OF ABBREVIATIONS

Abbreviation	Expansion
MVC	Model view controller
OO	Object oriented
SMS	Short Service Message
AT commands	Attention Commands
SMSC	Short Service Message center
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart

LIST OF FIGURES

Title	Page
Figure 2.1 MVC(model view control)	7
Figure 2.2 Onion Architecture	8
Figure 2.3 Leave management Design	10
Figure 3.1 Processing leave form	11
Figure 3.2 URL Routing	12
Figure 3.3 OO session handling	15
Figure 4.1 Login screen with Captcha	21
Figure 4.2 Adding Event only for Chief Warden	21
Figure 4.3 Applying Leaveform Page for the Student	22
Figure 4.4 Approval Status page for Student	22
Figure 4.5 Approval/Decline Page for Proctor	23

ABSTRACT

Leave Form Management facilitates students to apply leave online with the required authentication from the warden and proctor. This system creates an environment where concerned proctors and wardens are notified if the student applies for leave. The goal is to achieve a robust leave management of the students and their valuable time.

The existing system gives a paper form to fill the details of the student with the respective warden and proctor signature. The paper is so easy to lose and hard to keep record of the students. So this Leave Form Management records every detail in the database and easy to manage.

The Environment consists of students , wardens , proctors and CTS .The idea is to replace the existing system and paper. All the users of the system will be given a simple GUI(Graphical User Interface) to work on and simple 4-5 clicks to complete the job.

CHAPTER 1

INTRODUCTION

1.1 GENERAL

The aim is to make leave management online for the students, it also follows traditional procedure of approving by the concerned people and also notifying the parents/guardians of the student. It saves lots of paper and keep record of students whereabouts. The system can be remotely operated even with a mobile browser and get the job done only with a simple 4-5 clicks. In existing system facilitates student should fill a leave form and are required to get approved from concerned proctor and warden and gets a security check at the main gate of the campus, if everything is right he/she will be allowed to leave the campus. In this case the student record is hard to maintain and easy to lose the information.

1.2 MOTIVATION

In the literature study the student has to fill the offline paper form and get it signed from the concerned proctor and then by the warden of the hostel and then they are allowed to leave the campus. The offline system is very tedious and sometimes can be ever lasting.

1.3 PROBLEM DESCRIPTION

Leave Form Management facilitates students to apply leave online with the required authentication from the warden and proctor. This system creates an environment where concerned proctors and wardens are notified if the student applies for leave. The goal is to achieve a robust leave management of the students and their valuable time.

1.4 RELATED WORK

1.4.1 MVC(Model View Controller)

The Model:

In its simplest form the model stores data which is to be accessed by the view and written to by the controller.

The model is the most complex of all the parts of the system and will contain all the logic which is specific to the application and where domain entities which relate to real world concepts (such as "a user" or "an order") are stored. It is the part of the application which takes data (from any source) and processes it. The model also handles all data access and storage. It has no knowledge of any controllers or views which may use it.

For example, in PHP the model may represent a "User" in the system. It will handle any operations regarding users. Saving/loading records, validating registrations.

The model is not (common mistakes made by those misinterpreting the pattern):

- A simple data access point
- A class called "model" which represents a single database table

The View:

The view contains all the display logic. In PHP it will be the part of the application which generates the HTML. It has direct access to the Model and can query the model to get its data. It can create callbacks to its controller (for example a clicking a button in the view would trigger an action in the controller). A lot of MVC examples state that the view is decoupled from everything else and fed data by the controller. This is entirely inaccurate (see: Model-View-Confusion part 1: The View gets its own data from the Model for a detailed explanation). In MVC the views queries the model to request its own data.

The View is not (common mistakes made by those misinterpreting the pattern):

- Absent of logic
- Given data by the controller

The Controller:

The controller takes user input and updates the model where required. Where there is no user interaction (e.g. where a static data set is displayed and will be the same every time), no controller should be necessary. It is important to note that the controller is not a mediator or gateway between the view and the model. The view gets its own data from its model. The controller accesses the model but does not contain any display logic itself. All the controller does is respond to user input.

Each controller is linked to a single instance of a view and a single instance of a model.

The Controller is not (common mistakes made by those misinterpreting the pattern):

- A gateway/mediator between the model and the view.
- The place where views get initialised based on the state of a model. The controller is linked to a single view class (although it could be assigned to multiple instances) and responds to actions on it. For example a list of products would be a single view. The controller would handle user actions such as sorting the list, filtering the records it's displaying based on criteria specified by users. It would not also deal with displaying the info for a single product. This is a different view, which requires its own controller.

Program flow

The typical program flow in MVC is:

- The model, view and controller are initialised
- The view is displayed to the user, reading data from the model
- The user interacts with the view (e.g. presses a button) which calls a specified controller action
- The controller updates the model in some way
- The view is refreshed (fetching the updated data from the model)

1.4.2 Domain Model Mapper

What is it?

The Data Mapper is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.

Description:

This is a cleaner separation of concerns than that found in the ubiquitous Active Record pattern which, while a useful construct, conflates business logic with persistence. This can make a big difference in terms of testability as using a Data Mapper decouples the database from your domain models, making it easy to write unit tests. This has been something found slightly difficult with Django where the emphasis is more on writing integration tests that use fixtures to set up the test environment; writing unit tests without using a database is hard when foreign key constraints are involved.

1.5 SYSTEM REQUIREMENTS

Hardware Requirements:

- Server
- CPU and Memory
- Disk

Software Requirements:

- Leave From Management is developed on php ,mysql and runs on webserver.
- Operating System
Windows, MacOS, Linux, Solaris and just about anything that supports the required server software
- PHP
The web server must support PHP.
- Database
Mysql

1.6 REPORT ORGANIZATION

The report first summarizes the objective of the project then, it emphasizes on the related work. It then introduces the concept and explains understanding of the project. The report then describes the work done to achieve the objective. Then the further enhancements and conclusions are discussed.

CHAPTER 2

ANALYSIS AND DESIGN

2.1 INTRODUCTION TO PROBLEM DEFINITION

Leave Form Management facilitates students to apply leave online with the required authentication from the warden and proctor. This system creates an environment where concerned proctors and wardens are notified if the student applies for leave. The goal is to achieve a robust leave management of the students and their valuable time.

The existing system gives a paper form to fill the details of the student with the respective warden and proctor signature. The paper is so easy to lose and hard to keep record of the students. So this Leave Form Management records every detail in the database and easy to manage.

The Environment consists of students , wardens , proctors and CTS .The idea is to replace the existing system and paper. All the users of the system will be given a simple GUI(Graphical User Interface) to work on and simple 4-5 clicks to complete the job.

2.2 PROPOSED SYSTEM ARCHITETURE

Leave Form Management follows MVC(ModelViewControl) architecture in the abstract and Onion architecture in the Model layer. It is not just any other framework of web development. It has a full capability to support rapid web application development and dynamic interactivity with the database .

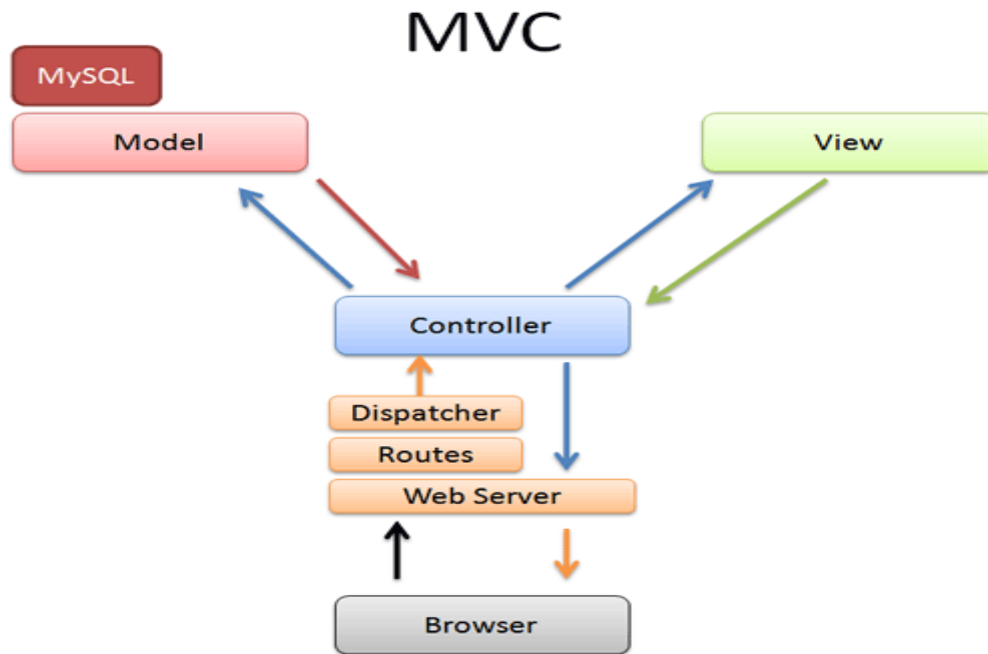


Figure 2.1 MVC(model view control)

A **controller** can send commands to the model to update the model's state (e.g., filling a the leave form). It can also send commands to its associated view to change the view's presentation of the model (e.g., view approved leave forms).

A **model** notifies its associated views and controllers when there has been a change instate. This notification allows views to update their presentation, and the controllers to change the available set of commands. In some cases an MVC implementation might instead be "passive," so that other components must poll the model for updates rather than being notified.

A **view** is told by the controller all the information it needs for generating an output representation to the user. It can also provide generic mechanisms to inform the controller of user input.

Onion Architecture:

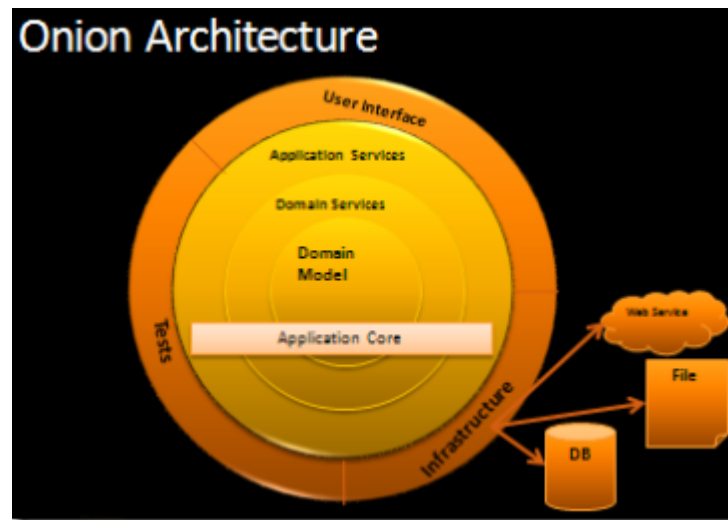


Figure 2.2 Onion Architecture

The diagram depicts the Onion Architecture. The main premise is that it controls coupling. The fundamental rule is that all code can depend on layers more central, but code cannot depend on layers further out from the core. In other words, all coupling is toward the center. In the very center we see the Domain Model, which represents the state and behavior combination that models truth for the organization. Around the Domain Model are other layers with more behavior. The number of layers in the application core will vary, but remember that the Domain Model is the very center, and since all coupling is toward the center, the Domain Model is only coupled to itself. The first layer around the Domain Model is typically where we would find interfaces that provide object saving and retrieving behavior, called repository interfaces. The object saving behavior is not in the application core, however, because it typically involves a database. Only the interface is in the application core. Out on the edges we see UI, Infrastructure, and Tests. The outer layer is reserved for things that change often. These things should be intentionally isolated from the application core. Out on the edge, we would find a class that implements a repository interface. This class is coupled to a particular method of data access, and that is why it resides outside the application core. This class implements the repository interface and is thereby coupled to it.

2.3 REQUIREMENT ANALYSIS

2.3.1 FUNCTIONAL REQUIREMENTS

2.3.1.1 USER REQUIREMENT

Students are provided with a from and to date calendar options where he could select the timings and date of his/her departue or leaving. On the other side Proctor/Warden will be able to approve with simple button.

Every input is validated for its purpose and irrelevant information is notified appropriately.

The system maintains history of the user details such as requested leave forms and any query will be redirected and fetches the results.

2.3.1.2 DOMAIN REQUIREMENT

Unauthorized access is strictly denied by the system and it has very high standard encryption implementation of password. Captcha is implemented and highly secured to break the captcha.

Entry of valid dates is mandatory since every process has input of date.

2.3.2 NON FUNCTIONAL REQUIREMENT

Performance requirements

Can handle the user request and operate on them simultaneously. The performance of the ssystem is not compromised for more users.

Operating constraints

It operates on a very tiny constraints and definitely has some minor constraints like support of php by the browser.

Platform constraints

Cross platform application since it is a web application, as it all needs is browser to operate the application

Accuracy and Precision

Definitely needed a suitable data to operate on and can not be partial.

Reliability

The application will be up and running as long as the servers are available.

Security

Captcha,session controllers for security and md5 algorithm for encryption.

Usability

Any user can operate the system as it is only clicks and very less data entry.

2.4 DETAILED DESIGN

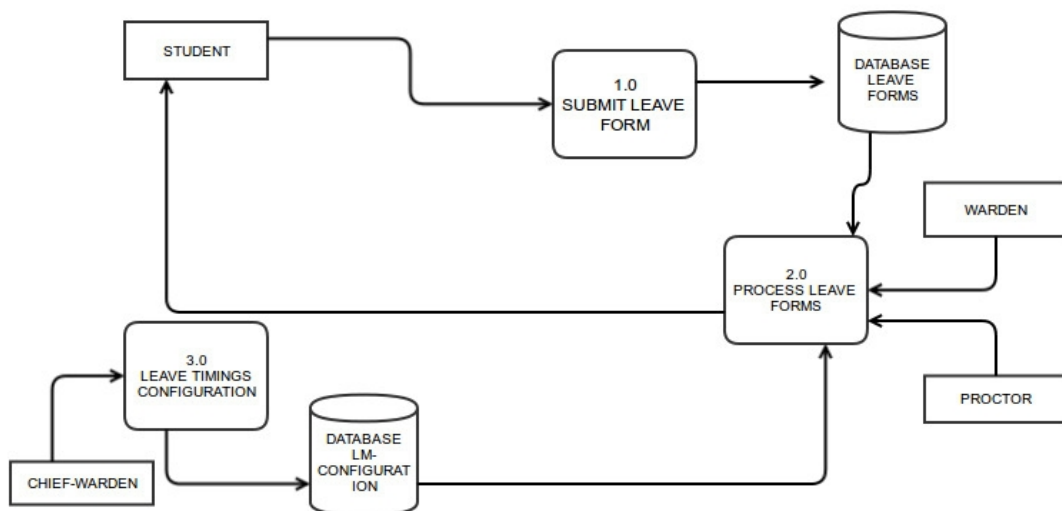


Figure 2.3 Leave management Design

This is overall application where students submit their leave form and they are forwarded to the warden and if necessary to proctor based on constraints(weekday,emergencies etc.,) and leaveforms are processed with the given configuration by the chief warden.

CHAPTER 3

IMPLEMENTATION

3.1 MODULES

3.1.1 Administration

The admin of this application will have the privilege to control all other functioning modules. The basic functions provided will be updating, monitoring the information. The admin will also be entertained to add, delete and update the users in the application.

3.1.2 User Management

The recognized users are student, wardens, proctor. Their interaction with the system will be logged into the system database, like profile management, leave details etc., will be saved for the future purpose.

3.1.3 Leave Form Processing

The soul of the project is to process the form. It starts from filling the form to approval of it by concerned warden and proctor. The notification of the student's leave to proctor and warden with the help of Email and SMS.

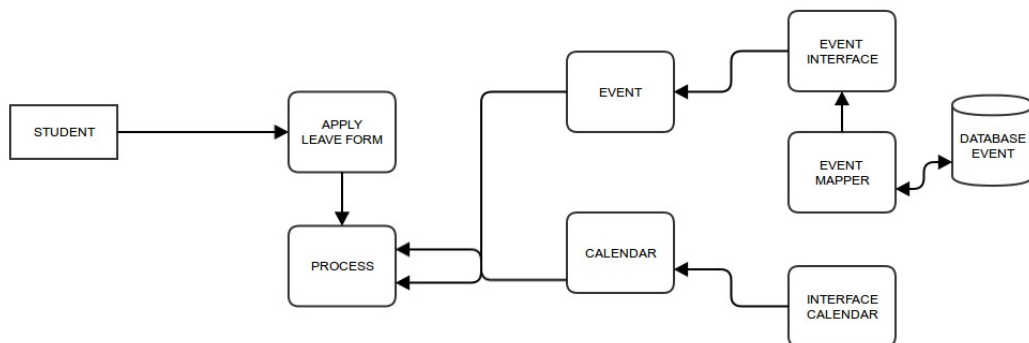


Figure 3.1 Processing leaveform

3.1.4 User Interface

The interface module is the face of the system, which allows the users to interact with the system. The interface is kept simple and provided with easy-to-use options and built using HTML5, bootstrap, cs, j Query and Java script.

3.2 TECHNICAL MODULES

3.2.1 URL Routing

Routing classes are typically used in conjunction to an MVC layout (Model-View-Controller) and it is responsible for mapping URL's to their corresponding Controller and delegate methods. You can use different HTTP methods to fully benefit from using REST / resourceful routes. This also allows you to create SEO friendly URL's that can also pass dynamic parameters to their corresponding Controller.

The main features typically are:

- Use different HTTP methods for same URL's to determine different types of action / request
- Support dynamic URL segments.
- Provide the capability for reverse routing; creating the destination URL from an array of route variables.

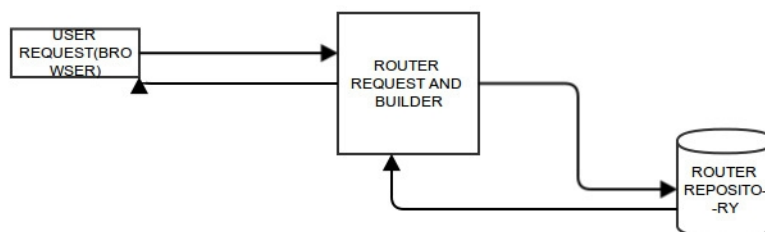


Figure 3.2 URL Routing

These classes are usually simple and lightweight and act as a fundamental point of any application. Below is some code put together to demonstrate how to create a routing class.

```
# Create our class
$router = new Router();
$router->setBasePath('/Router');

# Paths to map
$router->map('GET|POST','/', 'home#index', 'home');
$router->map('GET','/users/', array('c' => 'UserController', 'a' => 'ListAction'));
$router->map('GET','/users/[i:id]', 'users#show', 'users_show');
$router->map('POST','/users/[i:id]/[delete|update:action]', 'usersController#doAction',
'users_do');

# Generate the URL
$router->generate('users_show', array('id' => 5));
```

3.2.2 Dependency Injection

What is it?

Dependency injection is a software design pattern in which one or more dependencies (or services) are injected, or passed by reference, into a dependent object (or client) and are made part of the client's state. The pattern separates the creation of a client's dependencies from its own behavior, which allows program designs to be loosely coupled and to follow the dependency inversion and single responsibility principles. It directly contrasts the service locator pattern, which allows clients to know about the system they use to find dependencies.

Dependency injection involves four elements: the implementation of a service object; the client object depending on the service; the interface the client uses to communicate with the service; and the injector object, which is responsible for injecting the service into the client. The injector object may also be referred to as an assembler, provider, container, factory, or spring.

Implementation of dependency injection is often identical to that of the strategy pattern, but while the strategy pattern is intended for dependencies to be interchangeable throughout an object's lifetime, in dependency injection only a single instance of a dependency is used.

sample constructor injection

```
class Photo {  
    /**  
     * @var PDO The connection to the database  
     */  
    protected $db;  
  
    /**  
     * Construct.  
     * @param PDO $db_conn The database connection  
     */  
    public function __construct($dbConn)  
    {  
        $this->db = $dbConn;  
    }  
}
```

```
$photo = new Photo($dbConn);
```

Above, we're **injecting** all required dependencies, when the class's constructor method runs, rather than hardcoding them directly into the class.

3.2.3 Object Oriented Session Handling

Used OOP for handling sessions as it has lot of benefits, One of the promises of OOP is that you'll get code that closely models your problem domain and also the MVC controller asking the Request it is servicing which Session it belongs to, and what User is associated with that.

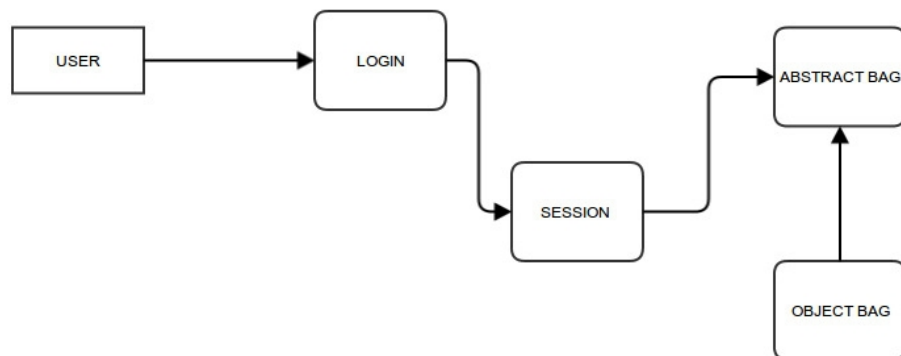


Figure 3.3 OO session handling

sample implementation

```
<?php  
class Session{
```

```

protected static $session_instance;
protected $session_data = array();
protected function __construct() { }
public static function getInstance(){
    if(empty(self::$session_instance)){
        self::$session_instance = new Session();
    }

    return self::$session_instance;
}

public function set($key, $val){
    $this->session_data[$key] = $val;
    return false;
}

public function get($key){
    if(!empty($this->session_data[$key])){
        return $this->session_data[$key];
    }
}

public function destroy($key){
    unset($this->session_data[$key]);
    return false;
}
}

/* Example */
$session = Session::getInstance();
$session->set('user_id', "203");
$user_id = $session->get('user_id');
$session->destroy('user_id');

```


3.2.4 Container Injection

A modern PHP application is full of objects. One object may facilitate the delivery of email messages while another may allow you to persist information into a database.

In your application, you may create an object that manages your product inventory, or another object that processes data from a third-party API. The point is that a modern application does many things and is organized into many objects that handle each task.

Dependency Injection is a design pattern where you pass dependencies to services instead of creating them from within the service or relying on globals. This generally leads to code that is decoupled, re-usable, flexible and testable. A DIC or service container is responsible for creating and storing services. It can recursively create dependencies of the requested services and inject them. It does so lazily, which means a service is only created when you actually need it. A service container is basically an array (or object) that contains other objects and sets default behaviors.

Sample Implementation

```
<?php
class Container implements ServiceInterface{

    protected $di;

    public function __construct(){
        $this->di = new AuraContainer(new Forge(new Config));
        $this->renderServices();
    }

    public function renderServices(){

        $instance = new \LeaveFormManagement\Persistence\DatabaseAdapter;
        $this->di->params['LeaveFormManagement\Session\Session'] = ['handler' => $this->di->lazyGet('handler')];
        $this->di->params['LeaveFormManagement\Mapper\AbstractDataMapper'] =
        ['adapter'=> $instance::getInstance() , 'collection' => $this->di->lazyGet('collection'), 'map'=> $this->di->lazyGet('EntityMap') ];
```

```

$this->di->params['LeaveFormManagement\Service\LoginForm'] = ['form' => $this->di-
>lazyGet('Form')];
    return $this;
}
public function get(){
    return $this->di;
}}

```

3.2.5 Encryption Standards

Used md5 to encrypt the information in session management and password storing.

Sample Implementation

```

<?php
include_once './EncrypterInterface.php';
class TextEncrypter implements EncrypterInterface{
    public function TextEncrypt($string) {
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256,
MCRYPT_MODE_CBC),MCRYPT_DEV_URANDOM);

        $encrypted = base64_encode(
            $iv .mcrypt_encrypt(
                MCRYPT_RIJNDAEL_256,
                hash('sha256', $key, true),
                $string,
                MCRYPT_MODE_CBC,
                $iv)
            );
        return $encrypted;
    }
}

```

3.2.6 Captcha

Generated a jpg image to determine whether or not the user is human.

Sample Implementation

```
<?php
    session_start();
    $random_alphanumeric = md5(rand());
    $captcha_code = substr($random_alphanumeric,0,6);

    $_SESSION["captcha_code"] = $captcha_code;
    $target_layer = imagecreatetruecolor(100,40);
    $captcha_background = imagecolorallocate($target_layer,255,160,119);

    imagefill($target_layer,5,5,$captcha_background);

    $captcha_text_color = imagecolorallocate($target_layer,0,0,0);
    imagestring($target_layer,10,5,5,$captcha_code,$captcha_text_color);

    header("content-type: image/jpeg");
    imagejpeg($target_layer);

?>
```

3.3 Unit TestCases

Test case Id	Test case Description	Expected Result	Actual Result	Remarks
1	User Authentication	Redirect to home page	PASS	
2	Leave form fields missing	Alert the user to enter the details	PASS	
3	Reload and misread captcha	Load another captcha and alert the user	PASS	
4	Invalid Dates for the leave	Alert saying invalid leave form dates	PASS	
5	Session creation after login	Session should be created for the user only after the login	PASS	
6	Update or Edit password	Change of password should be reflected in the database	PASS	
7	Email notification	A notification message should be mailed to the concerned	PASS	
8	SMS notification	Notification message should be sent to the concerned.	PASS	

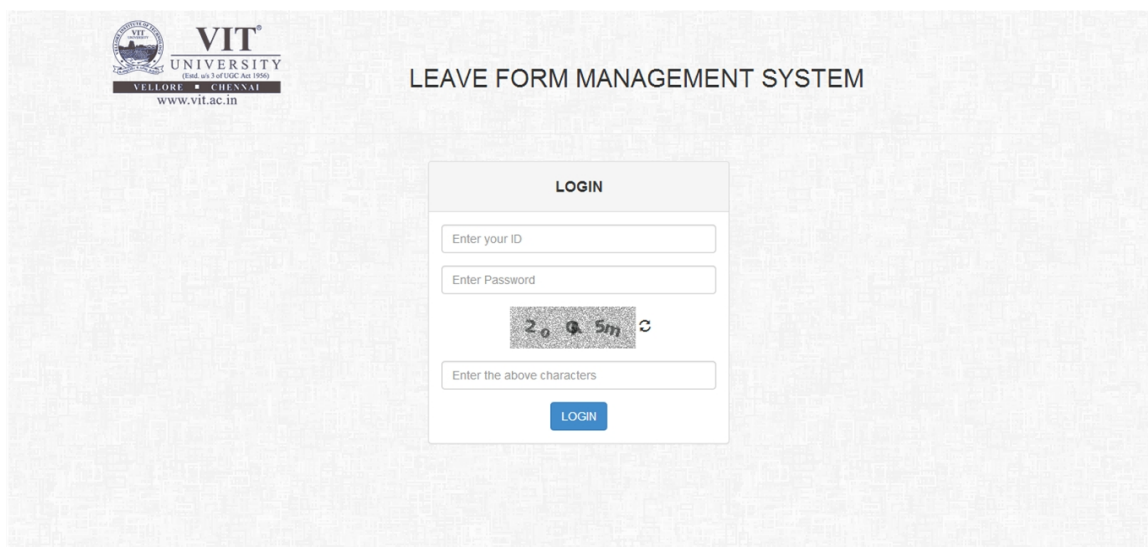
CHAPTER 4

RESULTS AND DISCUSSION

4.1 RESULTS

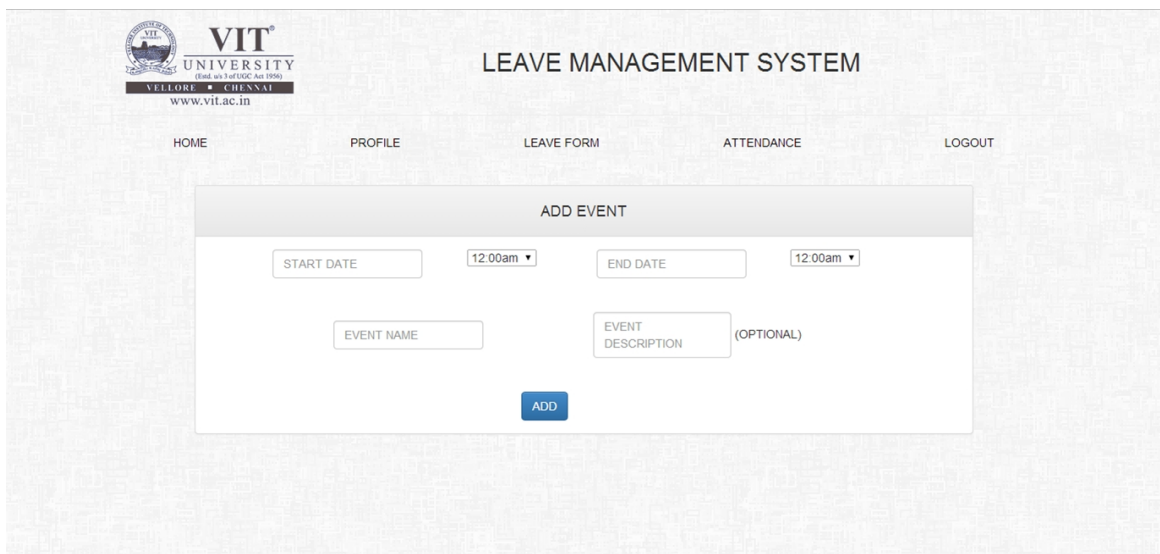
Students are able to apply leaveform online and it is forwarded to the concerned proctor and warden for the approval. A secured login with captcha wall for non human detection and encryption of required data with the encryption standards.

Sample screenshots



The screenshot shows the login interface of the VIT University Leave Form Management System. The header includes the VIT University logo and name, along with the website URL www.vit.ac.in. The main heading is "LEAVE FORM MANAGEMENT SYSTEM". The login form is titled "LOGIN" and contains the following fields: "Enter your ID", "Enter Password", a captcha image showing the characters "20 Q 5m", and "Enter the above characters". A blue "LOGIN" button is positioned at the bottom of the form.

Figure 4.1 Login screen with captcha



The screenshot displays the "ADD EVENT" form within the VIT University Leave Management System. The header shows the VIT University logo and name, and the website URL www.vit.ac.in. The main heading is "LEAVE MANAGEMENT SYSTEM". Below the header, there is a navigation bar with links: HOME, PROFILE, LEAVE FORM, ATTENDANCE, and LOGOUT. The "ADD EVENT" form is titled "ADD EVENT" and contains the following fields: "START DATE" (with a dropdown menu showing "12:00am"), "END DATE" (with a dropdown menu showing "12:00am"), "EVENT NAME", "EVENT DESCRIPTION" (marked as "OPTIONAL"), and an "ADD" button.

Figure 4.2 Adding Event* only for Chief Warden

Event

The days with the non-instructional day and custom holidays prescribed by the administration.

The screenshot shows the 'ADD EVENT' form in the VIT University Leave Management System. The form includes fields for 'START DATE', 'END DATE', 'EVENT NAME', and 'EVENT DESCRIPTION (OPTIONAL)'. Both date fields have a default time of '12:00am' selected from a dropdown menu. An 'ADD' button is located at the bottom of the form.

START DATE	12:00am	END DATE	12:00am
EVENT NAME		EVENT DESCRIPTION (OPTIONAL)	


ADD

Figure 4.3 Applying Leaveform Page for the Student

The screenshot shows the 'APPROVE LEAVE FORMS' page in the VIT University Leave Management System. It features a 'REFRESH' button and a table with columns for ID, NAME, PLACE, LEAVING TIME, RETURNING TIME, PURPOSE, and APPROVE/DECLINE. A single leave request for 'james' is listed with a status of 'PROCTOR APPROVAL PENDING'.

ID	NAME	PLACE	LEAVING TIME	RETURNING TIME	PURPOSE	APPROVE/DECLINE
10mse1025	james	home	2014-05-06 02:00:00	2014-05-06 03:00:00	festival	PROCTOR APPROVAL PENDING

Figure 4.4 Approval Status page for Student



VIT
 UNIVERSITY
(Est. as VIT-UGC Act 1996)
 VELLORE • CHENNAI
 www.vit.ac.in

LEAVE MANAGEMENT SYSTEM

[HOME](#)
[PROFILE](#)
[LEAVE FORM](#)
[VIEW STUDENT PROFILE](#)
[LOGOUT](#)

APPROVE LEAVE FORMS							
ID	NAME	PLACE	LAEAVING TIME	RETURNING TIME	PURPOSE	APPROVE/DECLINE	
10mse1102	S.CHAITHANIYA VARMA	KELAMBAKKAM,C HENNAI,TAMIL NADU	21-03-2014 9:00AM	21-03-2014 4:00PM	I AM GOING TO THE MOVIES	APPROVE	DECLINE

Figure 4.5 Approval/Decline Page for Proctor

CHAPTER 6

CONCLUSIONS AND FUTURE ENHANCEMENTS

Conclusion

Leaveform management online makes student lives better and easy and also many stakeholders of the system can do the job online. It also facilitates to operate remotely. It also gives you statistics of the students left, present in the hostels and gives information of the students whereabouts as it is online and stored in the database.

Notification of student when he/she leaves helps parents/guardians to keep track of their well beings.

Enhancement can be done including the attendance of the student in the Leave management which helps proctor to check the attendance and approve the leaveform.

REFERENCES

1. <http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
2. <http://jeffreypalermo.com/blog/the-onion-architecture-part-1/>
3. in2.php.net/manual/en/
4. www.php5dp.com/
5. <http://phpmvcframework.blogspot.in/2000/01/putting-it-altogether.html>