Introduction:

Property auction site for a regional estate agent office.

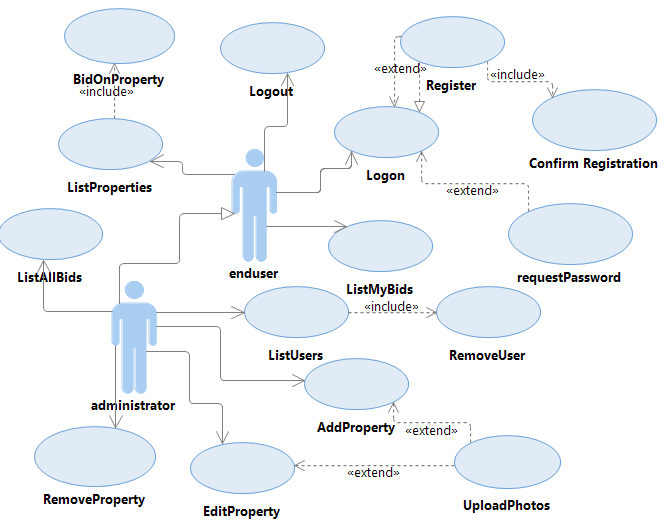
Refining the requirements:

Stakeholders require a website application that enables users to bid on various properties that they listed in a database. There will be two users of such a system the end user and the administrator. The end-user will be presented on log on with the logon screen in which they will enter the username and password. If they do not exist in the database, they can also request to register in which case they will be asked for their name and address Telephone number email address and password. They can also be presented with a forget password screen. When the user logs on to the site they will be presented with a list of properties. They should be able to view each one of these properties individually listing all the details that are to be confirmed at the moment. At the display of each property, the user can select to create a bid. In which case they will be presented with a minimum and maximum value that they wish to bid. If the world are bids already in the database they will be displayed to the user if their maximum value is less than any maximum value in the database. And the user will be shown their current bid value. The user will also be allowed to list all the bids they have active and to change their user details.

There will also be administrator user for this application that will be able to do everything that the user can, and various other methods that is he can list and modify all users, list all bids. List all properties, add a property edit property, and remove a property

Further modifications to these requirements will be added as this document develops.

Main Use Case Diagram:



Use Case

Looking at the above use Case 1:

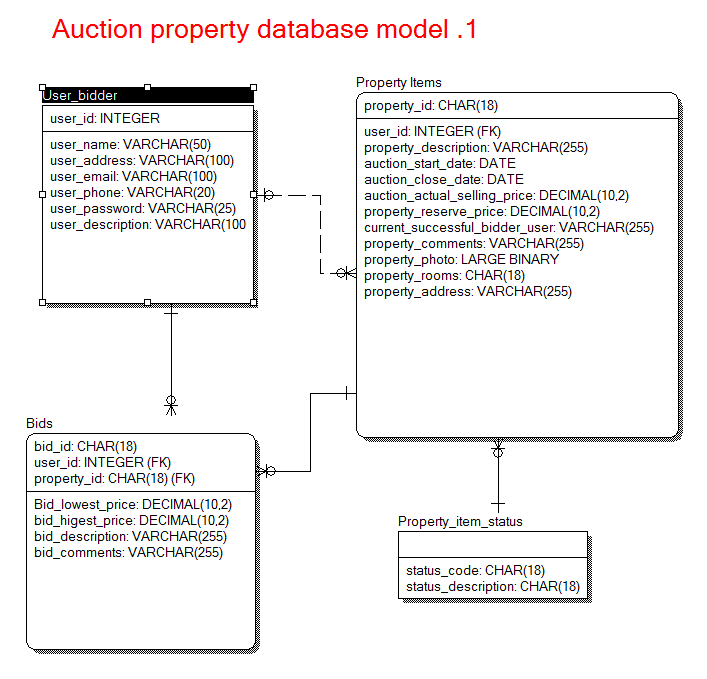
Actor enduser:

* Users can Logon to system  
  Request their password
* Register as a new user  
  Confirm Registration
* List Properties / Browse properties
* Bid on a property
* List user Bids

Actor Administrator extends enduser:

* List all Bids
* List all users
* List all property’s
* Modify user
* Remove user
* Add property
* Edit Property
* Remove Property
* Upload photos

MYSQL database and tables: ER Diagram design:



ER Diagram

As we can see from ER Diagram 1 there are four tables.

The bidder user that contains the user details including their password email address Telephone number e.c.t

The individual property items containing details of the property for auction start date the close date and the successful current bidder userid that is a foreign key for the user/bidder table

The property item status will contain a table of the status of the property. That could be either active non-active.

The bids table will contain a list of all bids on a particular property for a particular user. The table also contain the maximum and minimum bids for a particular property.

The cardinality between the tables is as follows a bidder/user can have zero one or many bids active. A bid must belong to one property and one bidder/user, whereas a property can have zero one or many bids.

TDD (Test-driven development)

This application will use test driven development and test units be created firstly.

List of URLs to be exposed by the API

this application will be designed in resource-Oriented Architecture (ROA) using rest the API is to be exposed by the system and the response calls are as follows: query string parameters are appropriate. Ie &content=xml

Response codes: 200 (OK), 401 (unauthorised), 304 (not modified), 500 (internal server error)

|  |  |  |
| --- | --- | --- |
| User |  |  |
| To add a new user: | post | http://localhost/user/ |
| To modify a user: | put | http://localhost/user/ID/ |
| To get a single user details: | get | http://localhost/user/ID/ |
| To Request a user password | get | http://localhost/user/ID/request |
|  |  |  |
| To get a list of all user bids | get | http://localhost/bids/userID/ |
| To add/create a new user bid: | post | http://localhost/bids/userID/ |
| To get a user individual bid details | get | http://localhost/bids/userID/BID\_ID/ |
| To modify this bid | put | http://localhost/bids/userID/BID\_ID/ |
| To delete this bid | delete | http://localhost/bids/userID/BID\_ID/ |
|  |  |  |
| Administrator |  |  |
| To get a list of all users | get | <http://localhost/user/> |
| To delete a user: | delete | <http://localhost/user/ID/> |
| List all Property | get | <http://localhost/Property> |
| Add new Property | post | <http://localhost/Property> |
| Remove Property | delete | <http://localhost/Property>/PROPID |
| Edit Property | put | <http://localhost/Property>/PROPID |
|  |  |  |
| To get a list of all bids | get | <http://localhost/bids> |
| To get a list of all bids for a particular property | get | <http://localhost/bids>/Admin/PROPID |
|  |  |  |

Data to be passed to and from the server API is to be formed both JSON and XML. Therefore two types of response: Example in JSON

|  |  |
| --- | --- |
| I.e. To get a single user details: | JSON &contenttype=json |
| Get http://localhost/user/10 | response  {  "response":"user\_bidder",  "user\_id":"10",  "user\_name":"username",  "user\_address":"the user address"  ...  } |
| I.e. To modify a user: | JSON to put |
| Put http://localhost/user/10 | {  "user\_id":"10",  "user\_name":"new username",  "user\_address":"new user address"  ...  } |

Folder structure of files

The Application MVC portfolio with Twitter Bootstrap

The application will rely on the above folder portfolio and will use the Twitter bootstrap

The application will be designed in an MVC architecture using MySQL as the database , but this database will be connected via a factory so it can be decoupled easily at any stage

High-level schema

The application will implement a three-tier validation system, given a robust validation scheme. Based on an MVC architecture strapped with Twitter and executing a data access factory. Implement the same design, as learnt in Web architectures class

The presentation tier will only display submit buttons when all other fields have been filled in i.e. in a logon screen the submit button will only become active when I username and password is entered in the correct format.

Server-side validation will validate if all information written or received the model are correct.

The data tier the database will have some validation rules as well applied to its input that are still to be defined.

|  |
| --- |
| Figure  High level schema |

Example reg expressions:

Validating an email string

function isEmailValid($emailStr){

$regex ="/([a-z0-9\_]+|[a-z0-9\_]+\.[a-z0-9\_]+)@(([a-z0-9]|[a- z0-9]+\.[a-z0-9]+)+\.([a-z]{2,4}))/i";

if(!preg\_match($regex ,$emailStr))return (false);

return (true);

}

Example view.php template

<?php

class View

{

private $model;

private $controller;

public function \_\_construct($controller,$model) {

$this->controller =$controller;

$this->model =$model;

}

public function output(){

$messageList =$this->model ->messageList;

$errorMessageSimpleForm =$this->model->errorMessageSimpleForm;

$errorMessageInteractiveForm =$this->model->errorMessageInteractiveForm;

//load the template file

include\_once($this->model->template);

}

}

?>

Example controller template: controller.php

<?php

class Controller{

private $model;

public function \_\_construct($model ,$action=null ,$parameters){

$this ->model =$model;

if($ction!=null) {

switch

($action) {

case "insertNewMessageFromSimpleForm":

$this ->insertNewMessageParameters($parameters , MESSAGE\_FORM\_SIMPLE);

break;

case "insertNewMessageFromInteractiveForm":

$this ->insertNewMessageParameters($parameters , MESSAGE\_FORM\_INTERACTIVE);

break;

default:

}

}

$this ->model ->prepareMessagesList();

}

public function insertNewMessageParameters($parameters ,$formType)

{

$authorEmail = $parameters["fAuthorEmail"];

$title =$parameters["fMessageTitle"];

$content = $parameters["fMessageContent"];

$areParsValid = $this ->model ->areMessageFormParametersValid($authorEmail ,$title ,$content);

if($areParsValid)

$this ->model ->insertNewMessage($authorEmail ,$title ,$content);

else

$this ->model ->setUpErrorMessage($formType);

}

}

?>

Example messages DAO

<?php

require\_once("dao.php");

class messagesDAO extends BaseDAO {

function messagesDAO($dbMng) {

parent::BaseDAO($dbMng);

}

function getuserss() {

$sqlQuery ="SELECT \*";

$sqlQuery .="FROM user";

$result =

$this ->getDbManager()->executeSelectQuery($sqlQuery);

return $result;

}

function insertNewUser($email ,$title ,$content) {

$sqlQuery = "INSERT INTO user (id,author,title,content)";

$sqlQuery .="VALUES(NULL,’$email’,’$title’,’$content’)";

$result = $this ->getDbManager()->executeQuery($sqlQuery);

return $result;

}

}

?>