Project Enhancements

Database

The Database layout was chosen to be in 3rd normal form to minimize duplication of data and safeguard against data anomalies (update, insertion and deletion anomalies)[1]. To still be able to use the database comfortably we decided to incorporate views[2]. Views are permanent, dynamic tables created from SELECT statements, and thus do not use any extra space in the database. The views allow the database to be a lot more efficient, as opposed to executing the same statement multiple times: the database engine automatically updates them as other queries are executed.

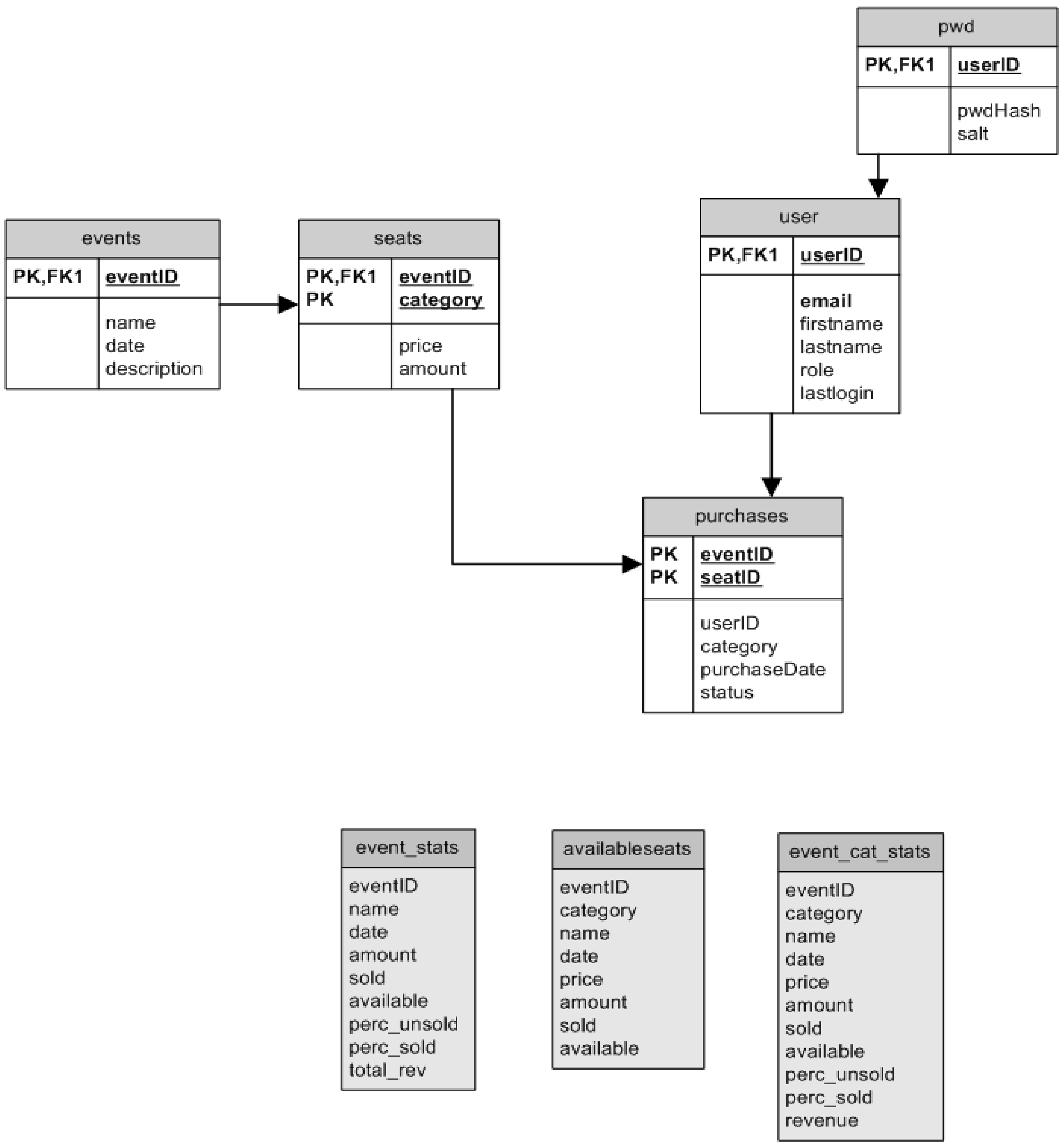
Furthermore our design allows us to enforce security measures such as storing the password table in a separate database (only accessible to the database admin) or providing reading allowances to end users for the statistical views (which do not contain any private or security relevant information). Currently the password is stored in a separate table in the same database, after being mixed with a random "salt" and then encrypted with sha1. The random salt is generated when registering the user, and is used to make it more difficult to brute-force passwords from the hash by using rainbowtables, or some other method.

The database user stored in our config.php has only the minimum rights necessary so can only insert datasets and issue select queries. This provides security in the case of security breach (sql-injection). That is, even if a user was to use SQL injection to break into the system, they could only view information and insert garbage into the tables, not delete entries or drop tables.

```
CREATE USER 'web'@'localhost' IDENTIFIED BY 'test';
GRANT SELECT,INSERT ON stss.* TO 'web'@'localhost';

CREATE VIEW availableseats AS
SELECT eventID, category, name, date, price, amount,
 COUNT(purchases.seatID) AS sold,
 (amount - COUNT(purchases.seatID)) AS available
  FROM events NATURAL LEFT OUTER JOIN seats
  NATURAL LEFT OUTER JOIN purchases
  GROUP BY eventID, category;
-- -----------------------------------------------------------
CREATE VIEW event_cat_stats AS
SELECT *,
 ((amount-sold)/amount*100) AS perc_unsold,
 ((sold)/amount*100) AS perc_sold,
 (price * sold) AS revenue
FROM `availableseats`;
-- -----------------------------------------------------------
CREATE VIEW event_stats AS
SELECT eventID, name, date,
 SUM(amount) AS amount,
 SUM(sold) AS sold,
 SUM(available) AS available,
 (SUM(available) / SUM(amount)*100) AS perc_unsold,
 (SUM(sold) / SUM(amount)*100) AS perc_sold,
 SUM(revenue) AS total_rev
FROM `event_cat_stats`
GROUP BY eventID;
```

1 http://en.wikipedia.org/wiki/Database_normalization
2 http://en.wikipedia.org/wiki/Database_view



Framework

We decided to implement our own Model View Controller (MVC) Framework to use in this project. Initially, progress moved slowly as we built up the structure of the framework. Once we got it up and running with an example that we could both follow, it proved to decrease development time by allowing changes and new features to be developed in parallel, and easily integrated together to form a cohesive website. For instance, one person would be working on improving the look of the website in the view only and the other enforcing sql-injection protection in the model. These simultaneous changes would not interfere with each other. The framework is made up of three parts, which we will discuss now:

CONTROLLER

```
<?php

//requires ...

//initSession
...

//Resolved the required module and action from GET
//defaults to default action
...

//sets session language
...

// initialize file paths to be used for requested action
$actionFile = BASE_PATH.'/modules/'.$module.'/'.$action.'.model.php';
//select View based on action and language settings
$viewFile = BASE_PATH.'/views/'.$_SESSION['lang'].'/'.$module.'/'.
$action.'.view.php';


//including model and view functions
...


//check if authenticated, included from model
      if (authenticate()) {

            //let the model do the work
            $result = work();

            //display the results with the view
            display($result);
      }
      ... no access

//if errors occured show here
...
?>
```

Index.php is our main component, the controller. It takes in (as GET parameters) the user's desired module and action to perform within that module. These are translated into file names, one for the "model" and one for the "view", which are then included into the page. The controller also initiates the session, selects the required language, and includes commonly used resources. Additionally, it enforces security by requiring an "authenticate" function to exist in every model, and will only display the results of the model if authentication was successful.

MODEL:

```
<?php

function authenticate () {
  //implement your own or call either
  //authNO();
  //or
  //authUser();
  //defined in Framework
  }

function work() {
  $result = array();
  //Do your work and
  //return $result = array();
  //with the results
  return $result;
}
?>
```

The model is responsible for handling all of the database queries and updating session variables where appropriate. It must implement an authenticate() function with is called in the controller. The work() function does the actual model state changes and returns a result in the form of an array.

VIEW:

```
<?php
function display($data) {
      showHeader(INDEX);

      //output here page content generated out of results stored
      //in array $data

      showFooter();
}
?>
```

In the view there is only one function called display(), which takes the $result array as a parameter. It is the responsibility of the view file to output the results in a nice fashion to the user.

Other Enhancements

- When purchasing a ticket, the user has the option to download the bill as a .pdf file.
- Events: Events may occur at any time on any day of any month of any year between 2008 and 2050. The administrator has the ability to add new events to the database.
- Switch between German and English, however database information is in one language only
- The business logic of the project is encapsulated within the views of the database - simply query the database and display the results of it.
- Using sql injection protection by binding parameters with mysqli or calling real_escape_string

Additional Technologies

- Scriptaculous/Prototype javascript library
- PEAR credit card processing
- ezPDF php library

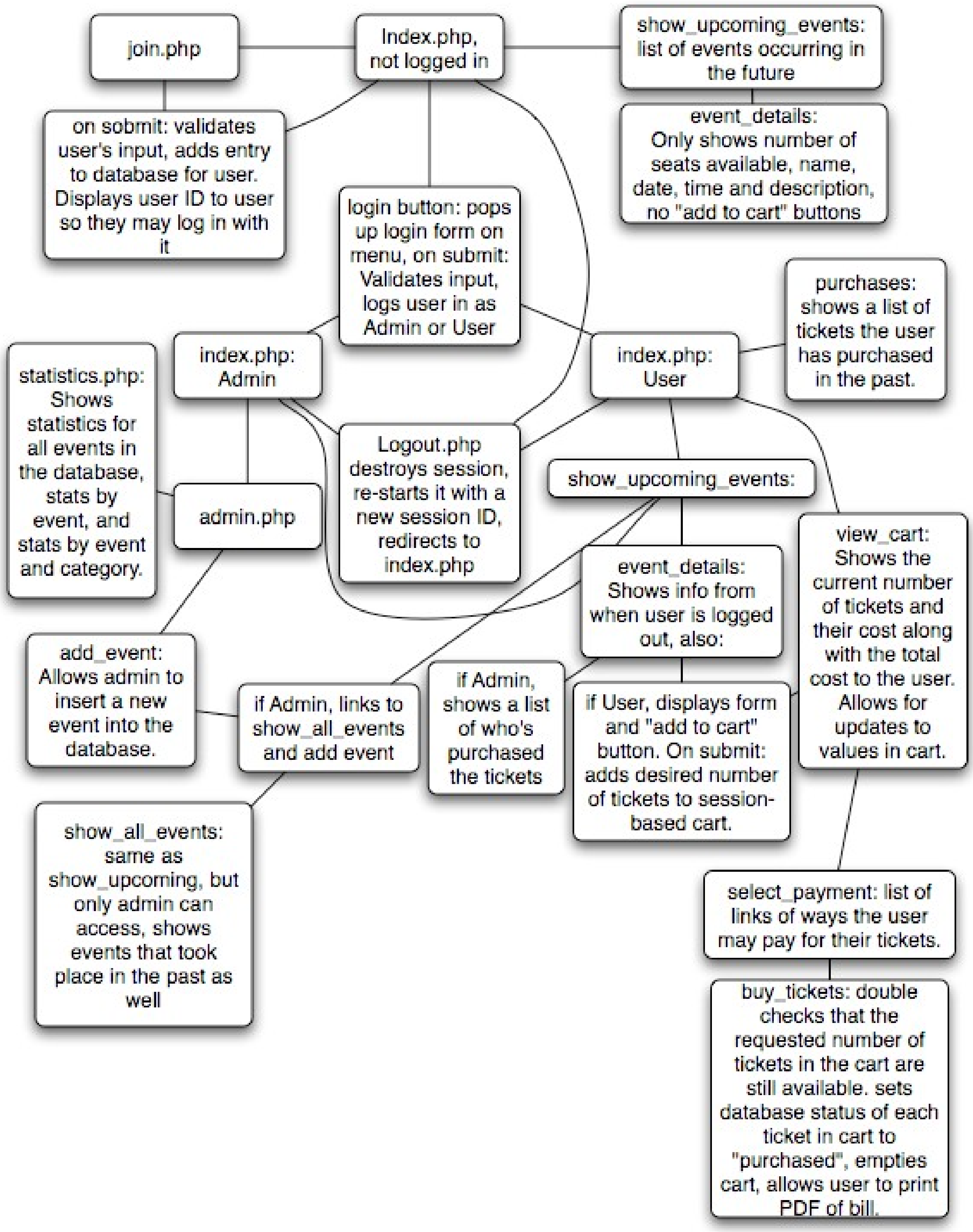
Known Vulnerabilities

- Not using ssl to encrypt network packet: sniffing network traffic could read the sessionID and hijack the session currently in progress.
- Cart is only stored in the session. Users could potentially fill their carts in parallel, and attempt to buy more tickets than are still available. This is checked for during the checkout process.
- Doesn’t validate as XHTML
- For simplicity’s sake, our framework is only loosely defined. We simply include files which have names the same as the arguments passed to the controller. If these files existed, then we assume that they have the proper functions defined in them, and simply call the functions to do the work. The results of the work are simply stored in an array which is passed to the view.
- The generated .pdf documents cannot be streamed directly to the user, so they are saved on the server’s hard drive, thus filling up the drive eventually. There should be an auto-cleanup script run on the server.

Poor design/User complaints:

- Doesn’t automatically log in after joining.
- Username/password boxes are backwards. True, but this way you can use <tab> to swap between them like normal. The reason behind this is because they’re “floated” using CSS to the right side of the page.

Site flowchart


join.php
Index.php, not logged in
show_upcoming_events: list of events occurring in the future
on sobmit: validates user's input, adds entry to database for user. Displays user ID to user so they may log in with it
event_details: Only shows number of seats available, name, date, time and description, no "add to cart" buttons
login button: pops up login form on menu, on submit: Validates input, logs user in as Admin or User
purchases: shows a list of tickets the user has purchased in the past.
statistics.php: Shows statistics for all events in the database, stats by event, and stats by event and category.
index.php: Admin
index.php: User
Logout.php destroys session, re-starts it with a new session ID, redirects to index.php
show_upcoming_events:
admin.php
view_cart: Shows the current number of tickets and their cost along with the total cost to the user. Allows for updates to values in cart.
event_details: Shows info from when user is logged out, also:
add_event: Allows admin to insert a new event into the database.
if Admin, links to show_all_events and add event
if Admin, shows a list of who's purchased the tickets
if User, displays form and "add to cart" button. On submit: adds desired number of tickets to session-based cart.
show_all_events: same as show_upcoming, but only admin can access, shows events that took place in the past as well
select_payment: list of links of ways the user may pay for their tickets.
buy_tickets: double checks that the requested number of tickets in the cart are still available. sets database status of each ticket in cart to "purchased", empties cart, allows user to print PDF of bill.

Folder Structure

- Webroot
  - Index.php
  - Config.php
  - Includes
  - Modules
    - Buy
    - Event
    - User
    - Welcome
  - Views
    - Css
    - Javascript
    - Images
    - Common.php
    - Language.php
    - Menu.php
    - De
      - Buy
      - Event
      - User
      - Welcome
    - En
      - Buy
      - Event
      - User
      - Welcome

Full Database Layout

```
CREATE USER 'web'@'localhost' IDENTIFIED BY 'test';

GRANT SELECT,INSERT ON stss.* TO 'web'@'localhost';

CREATE TABLE `events` (
  `eventID` int(11) NOT NULL auto_increment,
  `name` varchar(40) NOT NULL,
  `date` datetime NOT NULL,
  `description` text NOT NULL,
  PRIMARY KEY  (`eventID`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
-- -----------------------------------------------------------
CREATE TABLE `purchases` (
  `eventID` int(11) NOT NULL,
  `userID` int(11) NOT NULL,
  `seatID` int(11) NOT NULL,
  `category` varchar(40) NOT NULL,
  `purchaseDate` datetime NOT NULL,
  `status` varchar(40) default NULL,
  PRIMARY KEY  (`eventID`,`seatID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- -----------------------------------------------------------
CREATE TABLE `pwd` (
  `userID` int(11) NOT NULL,
  `pwdHash` varchar(256) NOT NULL,
  `salt` varchar(128) NOT NULL,
  PRIMARY KEY  (`userID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- -----------------------------------------------------------
CREATE TABLE `seats` (
  `eventID` int(11) NOT NULL,
  `category` varchar(40) NOT NULL,
  `price` float NOT NULL,
  `amount` int(11) NOT NULL,
  PRIMARY KEY  (`eventID`,`category`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- -----------------------------------------------------------
CREATE TABLE `user` (
  `userID` int(11) NOT NULL auto_increment,
  `email` varchar(40) NOT NULL,
  `firstname` varchar(40) NOT NULL,
  `lastname` varchar(40) NOT NULL,
  `role` tinyint(4) NOT NULL,
  `lastlogin` datetime NOT NULL,
  PRIMARY KEY  (`userID`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB;
-- -----------------------------------------------------------
CREATE VIEW availableseats AS
SELECT eventID, category, name, date, price, amount,
 COUNT(purchases.seatID) AS sold,
 (amount - COUNT(purchases.seatID)) AS available
  FROM events NATURAL LEFT OUTER JOIN seats
  NATURAL LEFT OUTER JOIN purchases
  GROUP BY eventID, category;
-- -----------------------------------------------------------
CREATE VIEW event_cat_stats AS
SELECT *,
 ((amount-sold)/amount*100) AS perc_unsold,
 ((sold)/amount*100) AS perc_sold,
 (price * sold) AS revenue
FROM `availableseats`;
-- -----------------------------------------------------------
CREATE VIEW event_stats AS
SELECT eventID, name, date,
 SUM(amount) AS amount,
 SUM(sold) AS sold,
 SUM(available) AS available,
 (SUM(available) / SUM(amount)*100) AS perc_unsold,
 (SUM(sold) / SUM(amount)*100) AS perc_sold,
 SUM(revenue) AS total_rev
FROM `event_cat_stats`
GROUP BY eventID;
```