```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Net.Mail;

using TARS.Models;
using TARS.Helpers;

namespace TARS.Controllers
{
    public class UserController : Controller
    {
        protected WorkEffortDBContext WorkEffortDB = new WorkEffortDBContext();
        protected HoursDBContext HoursDB = new HoursDBContext();
        protected TimesheetDBContext TimesheetDB = new TimesheetDBContext();
        protected TARSUserDBContext TARSUserDB = new TARSUserDBContext();
        protected DivisionsDBContext DivisionsDB = new DivisionsDBContext();
        protected EarningsCodesDBContext EarningsCodesDB = new EarningsCodesDBContext();
        protected PcaCodeDBContext PcaCodeDB = new PcaCodeDBContext();
        protected PCA_WEDBContext PCA_WEDB = new PCA_WEDBContext();
        protected HolidaysDBContext HolidaysDB = new HolidaysDBContext();

        //
        //Index view, redirects to viewTimesheet function
        [HttpGet]
        public virtual ActionResult Index()
        {
            Authentication auth = new Authentication();
            if (auth.isUser(this) || Authentication.DEBUG_bypassAuth )
            {
                return RedirectToAction("viewTimesheet", new { tsDate = DateTime.Now });
            }
            else
            {
                return View("notLoggedOn");
            }
        }


        //
        /* Adds the newhours object to the database. Before saving, it checks to see if the hours' timestamp
         * is within the time frame of the work effort that the hours are being logged to.
         */
        [HttpPost]
        public virtual bool addHours(Hours newhours)
        {
            Authentication auth = new Authentication();
            if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
            {
                WorkEffort tmpWe = WorkEffortDB.WorkEffortList.Find(newhours.workEffortID);

                //make sure that the new hours are within the work effort's time bounds
                if ((newhours.timestamp < tmpWe.startDate) || (newhours.timestamp > tmpWe.endDate))
                {
                    return false;
                }
                //make sure that a timesheet exists for the period hours are being added to
                checkForTimesheet(newhours.creator, newhours.timestamp);
                //add and save new hours
                HoursDB.HoursList.Add(newhours);
                HoursDB.SaveChanges();

                return true;
            }
            else
            {
                return false;
            }
        }


        //
        // Returns TRUE if the date is within the start and end dates of the the Work Effort
        public bool isWithinWeTimeBounds(int weID, DateTime hrsDate)
        {
            WorkEffort tmpWe = WorkEffortDB.WorkEffortList.Find(weID);
            if ((hrsDate >= tmpWe.startDate) && (hrsDate <= tmpWe.endDate))
            {
                return (true);
```

```
    }
    return false;
}


//
/* Checks if a timesheet exists for the specified user and date.
 * If not, then createTimesheet() is called
 */
[HttpGet]
public void checkForTimesheet(string userName, DateTime tsDate)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Timesheet resulttimesheet = new Timesheet();
        DateTime startDay = tsDate.StartOfWeek(DayOfWeek.Sunday);

        //Check if there is a timesheet for the week that corresponds to newhours.timestamp
        var searchTs = from m in TimesheetDB.TimesheetList
                       where (m.worker.CompareTo(userName) == 0)
                       where m.periodStart <= tsDate
                       where m.periodEnd >= tsDate
                       select m;
        foreach (var item in searchTs)
        {
            resulttimesheet = item;
        }

        //if there isn't a timesheet for the pay period, then create one
        //If there is a timesheet for the current pay period, don't do anything
        if (resulttimesheet.periodStart.CompareTo(startDay) != 0)
        {
            createTimesheet(userName, startDay);
            return;
        }
        return;
    }
    else
    {
        return;
    }
}


//
//Creates an empty timesheet for the specified user and specified date
public void createTimesheet(string userName, DateTime startDay)
{
    Timesheet newTimesheet = new Timesheet();
    var searchTs = from t in TimesheetDB.TimesheetList
                   where (t.worker.CompareTo(userName) == 0)
                   where t.periodStart == startDay
                   select t;
    try
    {
        //make sure the timesheet doesn't exist already
        if (searchTs.Count() == 0)
        {
            //Set pay period to start on Sunday 12:00am
            newTimesheet.periodStart = startDay;
            newTimesheet.periodEnd = startDay.AddDays(6.99999);
            newTimesheet.worker = userName;
            newTimesheet.approved = false;
            newTimesheet.locked = false;
            newTimesheet.submitted = false;

            //add timesheet and save to the database
            TimesheetDB.TimesheetList.Add(newTimesheet);
            TimesheetDB.Entry(newTimesheet).State = System.Data.EntityState.Added;
            TimesheetDB.SaveChanges();
        }
    }
    catch{}
}


//
//Retrieves and returns a specified user's timesheet for the specified date
public Timesheet getTimesheet(string user, DateTime tsDate)
{
    Timesheet resulttimesheet = new Timesheet();
```

```
    var searchTs = from m in TimesheetDB.TimesheetList
                   where (m.worker.CompareTo(user) == 0)
                   where m.periodStart <= tsDate
                   where m.periodEnd >= tsDate
                   select m;
    foreach (var item in searchTs)
    {
        resulttimesheet = item;
        return resulttimesheet;
    }
    return null;
}


//
/* Retrieves all workefforts from the database and sends them to the view,
 * along with a list of PCA Codes associated with each work effort
 */
[HttpGet]
public virtual ActionResult searchWorkEffort()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Authentication auth2 = new Authentication();
        if (auth2.isManager(this))
        {
            ViewBag.managerFlag = true;
        }

        var workEffortList = WorkEffortDB.WorkEffortList.ToList();
        //create a list of lists for pca codes
        //(each work effort will have a list of PCA codes)
        ViewBag.pcaListOfLists = new List<List<SelectListItem>>();
        foreach (var item in workEffortList)
        {
            ViewBag.pcaListOfLists.Add(getWePcaCodesSelectList(item));
        }

        return View(workEffortList);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves the Work Effort object with the specified ID and sends it to the view,
 * along with a list of associated PCA Codes.
 */
[HttpGet]
public virtual ActionResult viewWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Authentication newAuth = new Authentication();
        if (newAuth.isManager(this))
        {
            ViewBag.managerFlag = true;
        }

        WorkEffort workeffort = WorkEffortDB.WorkEffortList.Find(id);
        if (workeffort == null)
        {
            return HttpNotFound();
        }
        ViewBag.pcaList = getWePcaCodesSelectList(workeffort);
        ViewBag.WorkEffortID = workeffort.ID;
        return View(workeffort);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves the hours object with the specified ID, changes the number of hours,
 * and saves the changes to the database
```

```
 */
[HttpPost]
public virtual bool editHours(int id, int numHours)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Hours tmpHours = HoursDB.HoursList.Find(id);
        tmpHours.hours = numHours;
        //save changes to database
        HoursDB.Entry(tmpHours).State = EntityState.Modified;
        HoursDB.SaveChanges();
        return true;
    }
    else
    {
        return false;
    }
}


//
/* Removes all the zero hours entries for the given workEffort/timeCode pair for the
 * week on the user's timesheet. This results in that workEffort/timeCode pair being
 * removed from the viewTimesheet View. Redirects to viewTimesheet().
*/
[HttpPost]
public virtual ActionResult timesheetRemoveWorkEffort(int sundayID)
{
    Hours sunHours = HoursDB.HoursList.Find(sundayID);
    DateTime sunday = sunHours.timestamp;

    var searchHours = from h in HoursDB.HoursList
                      where (h.creator.CompareTo(sunHours.creator) == 0)
                      where h.workEffortID == sunHours.workEffortID
                      where (h.description.CompareTo(sunHours.description) == 0)
                      where h.timestamp >= sunday
                      where h.timestamp < System.Data.Objects.EntityFunctions.AddDays(sunday, 7)
                      select h;
    foreach (var item in searchHours)
    {
        HoursDB.HoursList.Remove(item);
        HoursDB.Entry(item).State = System.Data.EntityState.Deleted;
    }
    HoursDB.SaveChanges();
    return RedirectToAction("viewTimesheet", new { tsDate = sunday.AddDays(1) });
}


//
//Returns TRUE if the specified timesheet is locked
public bool isTimesheetLocked(string worker,DateTime refDate)
{
    DateTime todaysDate = DateTime.Now;
    Timesheet tmpTimesheet = getTimesheet(worker, refDate);
    if ( (tmpTimesheet != null)&&(tmpTimesheet.locked == true) )
    {
        return true;
    }
    return false;
}


//
// Retrieves the hours object with the specified ID and deletes it from the database
[HttpGet]
public virtual ActionResult deleteHours(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Hours hours = HoursDB.HoursList.Find(id);
        HoursDB.Entry(hours).State = EntityState.Deleted;
        HoursDB.SaveChanges();
        return RedirectToAction("viewTimesheet", new { tsDate = hours.timestamp });
    }
    else
    {
        return View("notLoggedOn");
    }
}
```

```
//
/* Duplicates all hours entries from previous week's timesheet and sets number of hours
 * to zero for all of them.
 */
[HttpGet]
public virtual ActionResult copyTimesheet()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string userName = User.Identity.Name;
        Timesheet previousTimesheet = new Timesheet();
        DateTime dayFromPrevPeriod = DateTime.Now;
        dayFromPrevPeriod = dayFromPrevPeriod.AddDays(-7);
        List<Hours> resultHours = new List<Hours>();

        //Select the timesheet from the previous pay period if it exists
        var searchTs = from m in TimesheetDB.TimesheetList
                       where (m.worker.CompareTo(userName) == 0)
                       where m.periodStart <= dayFromPrevPeriod
                       where m.periodEnd >= dayFromPrevPeriod
                       select m;
        foreach (var item in searchTs)
        {
            previousTimesheet = item;
        }
        //Iterate through each entry from previous week and duplicate it for this week
        var searchHours = from m in HoursDB.HoursList
                          where (m.creator.CompareTo(userName) == 0)
                          where m.timestamp >= previousTimesheet.periodStart
                          where m.timestamp <= previousTimesheet.periodEnd
                          select m;
        foreach (var item in searchHours)
        {
            resultHours.Add(item);
        }
        foreach (var copiedHours in resultHours)
        {
            copiedHours.hours = 0;
            copiedHours.timestamp = DateTime.Now.StartOfWeek(DayOfWeek.Sunday);
            addHours(copiedHours);
        }
        return RedirectToAction("viewTimesheet", new { tsDate = DateTime.Now });
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves a list of the current user's hours for the time period that tsDate falls within.
 * The list is saved in TempData[], then convertHoursForTimesheetView() is called.
 * convertHoursForTimesheetView() returns a list of TimesheetRow objects, which is sent to the view.
 */
public virtual ActionResult viewTimesheet(DateTime tsDate)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string userName = User.Identity.Name;
        checkForTimesheet(userName, tsDate);    //creates timesheet if it doesn't exist
        Timesheet timesheet = getTimesheet(userName, tsDate);
        Timesheet prevTimesheet = getTimesheet(userName, tsDate.AddDays(-7));

        ViewBag.timesheet = timesheet;
        //The View won't provide a link to previous timesheet unless it exists
        if (prevTimesheet == null)
        {
            ViewBag.noPreviousTimesheet = true;
        }
        //select all hours from the timesheet
        var hoursList = from m in HoursDB.HoursList
                        where (m.creator.CompareTo(userName) == 0)
                        where m.timestamp >= timesheet.periodStart
                        where m.timestamp <= timesheet.periodEnd
                        select m;

        TempData["hoursList"] = hoursList;
        //convert hoursList into a format that the view can use
        List<TimesheetRow> tsRows = convertHoursForTimesheetView();
```

```
        ViewBag.workEffortList = getVisibleWorkEffortSelectList(getUserDivision());
        return View(tsRows);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Uses TempData["hoursList"] (assigned by the calling function) to create a list of objects
 * that each contain number of hours for Sun-Sat for each workEffort/timeCode pairing.  The
 * list of objects is then returned to the calling function
 */
public List<TimesheetRow> convertHoursForTimesheetView()
{
    WorkEffort effort = new WorkEffort();
    string effortDescription = "";
    List<WorkEffort> workEffortList = new List<WorkEffort>();
    List<string> timeCodeList = new List<string>();
    List<string> effortAndCodeConcat = new List<string>();
    List<TimesheetRow> tsRowList = new List<TimesheetRow>();

    //TempData["hoursList"] was assigned in viewTimesheet() before calling this method
    IEnumerable<Hours> hoursList = (IEnumerable<Hours>)TempData["hoursList"];

    //create a list of workEffort/timeCode pairings for the pay period
    foreach (var item in hoursList)
    {
        timeCodeList.Add(item.description);
        effort = WorkEffortDB.WorkEffortList.Find(item.workEffortID);
        if (effort != null)
        {
            workEffortList.Add(effort);
            effortAndCodeConcat.Add(effort.description + "::::" + item.description);
        }
    }
    //remove duplicates from the list
    effortAndCodeConcat = effortAndCodeConcat.Distinct().ToList();

    //for each unique workEffort/timeCode pairing
    foreach (var effortAndCode in effortAndCodeConcat)
    {
        TimesheetRow tmpTsRow = new TimesheetRow();

        /* save the work effort description and time code, then remove from the front of the list.
         * In the process, any duplicate pairs are ignored and removed by comparing to effortAndCodeConcat
         */
        for (int count = 0; count < 100; count++)
        {
            try
            {
                if ((effortAndCode.Contains(workEffortList.First().description)) &&
                     (effortAndCode.Contains(timeCodeList.First())))
                {
                    tmpTsRow.weID = workEffortList.First().ID;
                    tmpTsRow.workeffort = workEffortList.First().description;
                    tmpTsRow.timecode = timeCodeList.First();
                    workEffortList.RemoveAt(0);
                    timeCodeList.RemoveAt(0);
                    break;
                }
                else
                {
                    workEffortList.RemoveAt(0);
                    timeCodeList.RemoveAt(0);
                }
            }
            catch
            {
            }
        }

        //for each hours entry in the pay period
        foreach (var tmpVal in hoursList)
        {
            effortDescription = getWeDescription(tmpVal.workEffortID);
            //if the hours entry belongs to the unique workEffort/timeCode pairing
            if ((effortAndCode.CompareTo(effortDescription + "::::" + tmpVal.description) == 0))
            {
                switch (tmpVal.timestamp.DayOfWeek.ToString())
                {
```

```
                    case ("Sunday"):
                        tmpTsRow.sunHours = tmpVal;
                        break;
                    case ("Monday"):
                        tmpTsRow.monHours = tmpVal;
                        break;
                    case ("Tuesday"):
                        tmpTsRow.tueHours = tmpVal;
                        break;
                    case ("Wednesday"):
                        tmpTsRow.wedHours = tmpVal;
                        break;
                    case ("Thursday"):
                        tmpTsRow.thuHours = tmpVal;
                        break;
                    case ("Friday"):
                        tmpTsRow.friHours = tmpVal;
                        break;
                    case ("Saturday"):
                        tmpTsRow.satHours = tmpVal;
                        break;
                    default:
                        break;
                }
            }
        }
        //Add the TimesheetRow so it will be displayed in viewTimesheet View
        tsRowList.Add(tmpTsRow);
    }
    return tsRowList;
}


//
/* Retrieves and returns list of hours logged to the specified work effort during the specified
 * time frame by the specified user.
 */
public virtual ActionResult showWorkOnWorkEffort(int we, DateTime start, DateTime end, string userName)
{
    if (Request.IsAjaxRequest())
    {
        var hrsList = from h in HoursDB.HoursList
                      where (h.creator.CompareTo(userName) == 0)
                      where h.workEffortID == we
                      where h.timestamp >= start
                      where h.timestamp <= end
                      select h;
        return PartialView("_showWorkOnWorkEffort", hrsList.ToList());
    }
    return null;
}


//
/* Retrieves timesheet object with specified ID and changes submitted status to TRUE.
 * It then saves the change and redirects to viewTimesheet()
 */
public virtual ActionResult submitTimesheet(int id)
{
    if (id >= 0)
    {
        Authentication auth = new Authentication();
        if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
        {
            Timesheet ts = new Timesheet();
            ts = TimesheetDB.TimesheetList.Find(id);
            ts.submitted = true;
            //save changes to the database
            TimesheetDB.Entry(ts).State = System.Data.EntityState.Modified;
            TimesheetDB.SaveChanges();

            //send an email to employee to confirm timesheet submittal
            string body = "Your IDHW timesheet for the pay period of " + ts.periodStart +
                          " - " + ts.periodEnd + " has successfully been submitted.";
            SendEmail(ts.worker, "Timesheet Submitted", body);

            return RedirectToAction("viewTimesheet", new { tsDate = ts.periodStart.AddDays(2) });
        }
        else
        {
            return View("notLoggedOn");
        }
    }
```

```
    else
    {
        return View("error");
    }
}


//
/* Retrieves the specified employee's timesheet from the specified date and returns the
 * status as a string
 */
public virtual string getTimesheetStatus(string userName, DateTime refDate)
{
    Timesheet tmptimesheet = getTimesheet(userName, refDate);
    string status = "";

    if (tmptimesheet != null)
    {
        if (tmptimesheet.locked)
        {
            status = "locked";
        }
        else if (tmptimesheet.approved)
        {
            status = "approved";
        }
        else if (tmptimesheet.submitted)
        {
            status = "submitted";
        }
        else
        {
            status = "not submitted";
        }
    }
    return status;
}


//
//Returns the pay period for the reference date as a string
public virtual string getPayPeriod(DateTime refDate)
{
    DateTime startDay = refDate.StartOfWeek(DayOfWeek.Sunday);
    DateTime endDay = startDay.AddDays(6);
    string payPeriod = startDay.ToShortDateString() + " - " + endDay.ToShortDateString();
    return payPeriod;
}


//
// Retrieves the IDHW Divisions and returns as a list of strings
public virtual List<SelectListItem> getDivisionSelectList()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        List<SelectListItem> divList = new List<SelectListItem>();
        var searchDivisions = from m in DivisionsDB.DivisionsList
                              select m;

        divList.Add(new SelectListItem { Text = "All", Value = "All" });

        foreach (var item in searchDivisions)
        {
            divList.Add(new SelectListItem
            {
                Text = item.divName,
                Value = item.divName
            });
        }
        return divList;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves all PCA codes associated with the specified work effort and returns
 * them as a selection list
```

```
 */
public virtual List<SelectListItem> getWePcaCodesSelectList(WorkEffort we)
{
    List<SelectListItem> pcaList = new List<SelectListItem>();
    PcaCode tmpPca = new PcaCode();

    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == we.ID
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        tmpPca = PcaCodeDB.PcaCodeList.Find(item.PCA);
        pcaList.Add(new SelectListItem
        {
            Text = tmpPca.code + " (" + tmpPca.division + ")",
            Value = tmpPca.code.ToString()
        });
    }
    return pcaList;
}


//
/* Retrieves all PCA codes associated with the specified work effort and
 * returns them in a string, separated by commas
 */
public virtual string getWePcaCodesString(int weID)
{
    PcaCode tmpPca = new PcaCode();
    string pcaString = "";
    WorkEffort we = WorkEffortDB.WorkEffortList.Find(weID);

    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == we.ID
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        tmpPca = PcaCodeDB.PcaCodeList.Find(item.PCA);
        pcaString = pcaString + " " + tmpPca.code + ",";
    }
    pcaString = pcaString.TrimEnd(',');
    return pcaString;
}


//
/* Retrieves all Earnings Code objects, then creates and returns them as a list of
 * strings, each containing a concatenated earnings code and description.
 */
public virtual List<string> getTimeCodeList()
{
    List<string> timeCodesList = new List<string>();
    var searchEarnCodes = from m in EarningsCodesDB.EarningsCodesList
                          select m;
    timeCodesList.Add("--- Choose a Time Code ---");
    foreach (var item in searchEarnCodes)
    {
        timeCodesList.Add(item.earningsCode + " " + item.description);
    }
    return timeCodesList;
}


//
/* Retrieves all non-hidden Work Efforts within the specified division, then returns
 * them as a selection list. Since the division is not stored with a Work Effort, the
 * PCA Codes associated with each Work Effort instance must be retrieved.  If at
 * least one PCA Code is from the specified division, then the Work Effort is added
 * to the selection list.
 */
public virtual List<SelectListItem> getVisibleWorkEffortSelectList(string division)
{
    List<SelectListItem> effortList = new List<SelectListItem>();
    PcaCode tmpPca = new PcaCode();
    string tmpValue = "";

    var searchEfforts = from m in WorkEffortDB.WorkEffortList
                        select m;
    //narrow down to work efforts in the specified division
    //(PCA codes and PCA_WE must be used to get all work efforts in the division)
    foreach (var we in searchEfforts)
```

```
    {
        if (we.hidden != true)
        {
            var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                              where p.WE == we.ID
                              where p.active == true
                              select p;
            foreach (var pca_we in searchPcaWe)
            {
                tmpPca = PcaCodeDB.PcaCodeList.Find(pca_we.PCA);
                //if the PCA is in the user's division, then add the Work Effort to the list
                if (tmpPca.division.CompareTo(division) == 0)
                {
                    tmpValue = we.ID.ToString();
                    effortList.Add(new SelectListItem
                    {
                        Text = we.description,
                        Value = tmpValue
                    });
                    break;
                }
            }
        }
    }
    return effortList;
}


//
/* Retrieves the Work Effort object with specified ID and returns the description of
 * as a string.  If the Work Effort has been deactivated, then a message is returned
 * that shows the ID and states that the Work Effort no longer exists.
 */
public virtual string getWeDescription(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string weDescription = "";
        var searchWorkEfforts = from w in WorkEffortDB.WorkEffortList
                                where w.ID == id
                                select w;
        foreach (var item in searchWorkEfforts)
        {
            weDescription = item.description;
        }
        //If the Work Effort no longer exists, return the id and a message
        if (weDescription.Length == 0)
        {
            weDescription = "(Work Effort no longer exists. The unique ID was " + id + ")";
        }
        return weDescription;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves Work Effort object with specified ID and returns it's time
 * boundaries as a string (note: it's called from addHours View)
 */
public string getWeTimeBoundsString(int id)
{
    WorkEffort we = WorkEffortDB.WorkEffortList.Find(id);
    string bounds = we.startDate + " - " + we.endDate;
    return bounds;
}


//
// Retrieves the division that the logged in user works for and returns it as a string
public virtual string getUserDivision()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string division = "";
        var searchUsers = from m in TARSUserDB.TARSUserList
                          where (m.userName.CompareTo(User.Identity.Name) == 0)
                          select m;
```

```
        foreach (var item in searchUsers)
        {
            division = item.company;
        }
        return division;
    }
    else
    {
        return null;
    }
}


//
// Retrieves the department that the logged in user works for and returns it as a string
public virtual string getUserDepartment()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string department = "";
        var searchUsers = from m in TARSUserDB.TARSUserList
                          where (m.userName.CompareTo(User.Identity.Name) == 0)
                          select m;
        foreach (var item in searchUsers)
        {
            department = item.department;
        }
        return department;
    }
    else
    {
        return null;
    }
}


//
/* Returns title that is shown when hovering over a timesheet day cell in
 * viewTimesheet view. If the timesheet is submitted or locked, then the title
 * will reflect that (unless user is an Admin).  Otherwise, it will show
 * "Add/Edit Hours"
 */
public virtual string getTimesheetDayCellTitle(Timesheet ts)
{
    string title = "";
    Authentication auth = new Authentication();

    if ( (ts.approved == true) || (ts.locked == true) )
    {
        if (auth.isAdmin(this))
        {
            title = "Admin: Add/Edit Hours";
        }
        else
        {
            title = getTimesheetStatus(ts.worker, ts.periodStart);
        }
    }
    else
    {
        title = "Add/Edit Hours";
    }
    return title;
}


//
/* Calls getVisibleWorkEffortSelectList(), then converts the result into a Json list
 * and returns it
 */
public ActionResult jsonWorkEffortSelectList(string division)
{
    List<SelectListItem> weSelectList = getVisibleWorkEffortSelectList(division);
    List<string> weIDList = new List<string>();
    foreach (var item in weSelectList)
    {
        weIDList.Add(item.Value);
    }
    return Json(weIDList.Select(x => new { value = x, text = getWeDescription(Convert.ToInt32(x)) }),
                JsonRequestBehavior.AllowGet
                );
}
```

```
        //
        // Calls getTimeCodeListList(), then converts the result into a Json list and returns it.
        public JsonResult jsonTimeCodeSelectList()
        {
            IEnumerable<string> weSelectList = getTimeCodeList();

            return Json(weSelectList.Select(x => new { value = x, text = x }),
                        JsonRequestBehavior.AllowGet
                        );
        }


        //
        // Sends a reminder email to all users who haven't submitted their timesheet by Saturday morning
        public void reminderToSubmitTimesheet()
        {
            DateTime refDate = DateTime.Now;
            string body = "Please submit your IDHW timesheet by the end of the day today.  <br /><br />Thanks!";
            var searchTimesheets = from t in TimesheetDB.TimesheetList
                                   where t.periodStart <= refDate
                                   where t.periodEnd >= refDate
                                   where t.submitted != true
                                   select t;
            foreach (var item in searchTimesheets)
            {
                SendEmail(item.worker, "Reminder to submit timesheet today", body);
            }
            return;
        }


        //
        //Sends an email from local server
        internal static void SendEmail(string userName, string subject, string body)
        {
string toAddress = "zeke_long@hotmail.com";
            //string toAddress = getEmailAddress(userName);
            try
            {
                MailMessage mailMessage = new MailMessage();
                mailMessage.To.Add(new MailAddress(toAddress));
                mailMessage.Subject = subject;
                mailMessage.Body = body;
                mailMessage.IsBodyHtml = true;

                var client = new SmtpClient();
                client.Send(mailMessage);
            }
            catch (Exception ex)
            {
                var tmpVar = ex;
            }
        }

    }
}
```