public IActionResult DownloadSalesReceipt()

{

//Get the placements

var householdInfo = \_context.Members

.Include(m => m.Household)

.Include(m => m.Order).ThenInclude(m => m.OrderItem)

.AsEnumerable()

.OrderBy(m => m.Order.Date) //order by purchase date

.GroupBy(a => new { a.Household, a.Order, a.Order.OrderItem, a.Order.OrderItem.Item })

.Select(grp => new Sales

{

Household = grp.Key.Household.HCode,

PurchaseDate = grp.Key.Order.Date, //purchase date

Purchases = grp.Key.Item.Name + " (" + grp.Key.OrderItem.Quantity.ToString() + ")", //purchases and quantity

Taxes = grp.Key.Order.Taxes,

Total = grp.Key.Order.Total, //purchase total

Volunteer = grp.Key.Household.Members.ToString() //volunteer

});

//How many rows?

int numRows = householdInfo.Count();

if (numRows > 0) //We have data

{

//Create a new spreadsheet from scratch.

using (ExcelPackage excel = new ExcelPackage())

{

//Note: you can also pull a spreadsheet out of the database if you

//have saved it in the normal way we do, as a Byte Array in a Model

//such as the UploadedFile class.

//

// Suppose...

//

// var theSpreadsheet = \_context.UploadedFiles.Include(f => f.FileContent).Where(f => f.ID == id).SingleOrDefault();

//

// //Pass the Byte[] FileContent to a MemoryStream

//

// using (MemoryStream memStream = new MemoryStream(theSpreadsheet.FileContent.Content))

// {

// ExcelPackage package = new ExcelPackage(memStream);

// }

var workSheet = excel.Workbook.Worksheets.Add("Household Income Information");

//Note: Cells[row, column]

workSheet.Cells[3, 1].LoadFromCollection(householdInfo, true);

//Note: You can define a BLOCK of cells: Cells[startRow, startColumn, endRow, endColumn]

//Make Date and Athlete Bold

workSheet.Cells[4, 1, numRows + 2, 1].Style.Font.Bold = true;

//Note: these are fine if you are only 'doing' one thing to the range of cells.

//Otherwise you should USE a range object for efficiency

using (ExcelRange totals = workSheet.Cells[numRows + 6, 2])

{

totals.Formula = "Sum(" + workSheet.Cells[4, 6].Address + ":" + workSheet.Cells[numRows + 3, 6].Address + ")";

totals.Style.Font.Bold = true;

}

workSheet.Cells[numRows + 5, 1].Value = "Total Number of Athletes";

workSheet.Cells[numRows + 5, 1].Style.Font.Bold = true;

workSheet.Cells[numRows + 5, 2].Value = numRows;

workSheet.Cells[numRows + 5, 2].Style.Font.Bold = true;

workSheet.Cells[numRows + 6, 1].Value = "Total Number of Events";

workSheet.Cells[numRows + 6, 1].Style.Font.Bold = true;

//Set Style and backgound colour of headings

using (ExcelRange headings = workSheet.Cells[3, 1, 3, 6])

{

headings.Style.Font.Bold = true;

var fill = headings.Style.Fill;

fill.PatternType = ExcelFillStyle.Solid;

fill.BackgroundColor.SetColor(Color.LightBlue);

}

////Boy those notes are BIG!

////Lets put them in comments instead.

///use for product and quantity

for (int i = 4; i < numRows + 4; i++)

{

using (ExcelRange Rng = workSheet.Cells[i, 7])

{

string[] commentWords = Rng.Value.ToString().Split(' ');

Rng.Value = commentWords[0] + "...";

//This LINQ adds a newline every 7 words

string comment = string.Join(Environment.NewLine, commentWords

.Select((word, index) => new { word, index })

.GroupBy(x => x.index / 7)

.Select(grp => string.Join(" ", grp.Select(x => x.word))));

ExcelComment cmd = Rng.AddComment(comment, "Dietary Restrictions");

cmd.AutoFit = true;

}

}

//Autofit columns

workSheet.Cells.AutoFitColumns();

//Note: You can manually set width of columns as well

//workSheet.Column(7).Width = 10;

//Add a title and timestamp at the top of the report

workSheet.Cells[1, 1].Value = "Household Income Report";

using (ExcelRange Rng = workSheet.Cells[1, 1, 1, 6])

{

Rng.Merge = true; //Merge columns start and end range

Rng.Style.Font.Bold = true; //Font should be bold

Rng.Style.Font.Size = 18;

Rng.Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

}

//Since the time zone where the server is running can be different, adjust to

//Local for us.

DateTime utcDate = DateTime.UtcNow;

TimeZoneInfo esTimeZone = TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard Time");

DateTime localDate = TimeZoneInfo.ConvertTimeFromUtc(utcDate, esTimeZone);

using (ExcelRange Rng = workSheet.Cells[2, 6])

{

Rng.Value = "Created: " + localDate.ToShortTimeString() + " on " +

localDate.ToShortDateString();

Rng.Style.Font.Bold = true; //Font should be bold

Rng.Style.Font.Size = 12;

Rng.Style.HorizontalAlignment = ExcelHorizontalAlignment.Right;

}

//Ok, time to download the Excel

//I usually stream the response back to avoid possible

//out of memory errors on the server if you have a large spreadsheet.

//NOTE: Since .NET Core 3 most Web Servers disallow sync IO so we

//need to temporarily change the setting for the server.

//If we can't then we will try to build the file and return a FileContentResult

var syncIOFeature = HttpContext.Features.Get<IHttpBodyControlFeature>();

if (syncIOFeature != null)

{

syncIOFeature.AllowSynchronousIO = true;

using (var memoryStream = new MemoryStream())

{

Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

Response.Headers["content-disposition"] = "attachment; filename=HouseholdIncomeReport.xlsx";

excel.SaveAs(memoryStream);

memoryStream.WriteTo(Response.Body);

}

}

else

{

try

{

Byte[] theData = excel.GetAsByteArray();

string filename = "HouseholdIncomeReport.xlsx";

string mimeType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

return File(theData, mimeType, filename);

}

catch (Exception)

{

return NotFound();

}

}

}

}

return NotFound();

}