```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CupPlaner.Controllers;
using System.Web.Mvc;
using System.Data.Entity;

namespace CupPlaner.Helpers
{
    public class MatchGeneration
    {
        public static int matchNumber;
        //this function generates the group stages for a tournament
        public bool GenerateGroupStage(int tournamentID)
        {
            CupDBContainer db = new CupDBContainer();
            Tournament t = db.TournamentSet.Find(tournamentID);
            matchNumber = 1;

            foreach (Division d in t.Divisions)
            {
                //new divisiontounament for each division
                DivisionTournament dt = db.DivisionTournamentSet.Add(new
DivisionTournament() { TournamentStructure = d.TournamentStructure, Division = d });
                foreach (Pool p in d.Pools)
                {
                    //tournamentstage with the timeinterval set to go from the start of
the tournament to the end of the hole tournament
                    TournamentStage ts = db.TournamentStageSet.Add(new TournamentStage()
{ Pool = p, DivisionTournament = dt, TournamentStructure =
TournamentStructure.RoundRobin, TimeInterval = new TimeInterval() { StartTime =
t.TimeIntervals.First().StartTime, EndTime = t.TimeIntervals.Last().EndTime } });
                    List<Team> teams = p.Teams.ToList();
                    //each team gets set up against each other team once
                    for (int i = 0; i < teams.Count; i++)
                    {
                        for (int j = i + 1; j < teams.Count; j++)
                        {

                            db.MatchSet.Add(new Match() { Teams = { teams[i], teams[j] },
TournamentStage = ts, Duration = d.MatchDuration, Number = matchNumber++ });
                        }
                    }
                }
            }
            db.SaveChanges();
            return true;
        }

        //this function generates the teams and pools needed for the finalstage
        public bool GenerateFinalsTeams(int tournamentID)
        {
            CupDBContainer db = new CupDBContainer();
            Tournament t = db.TournamentSet.Find(tournamentID);
            foreach (Division d in t.Divisions)
            {
```

```
                int finalsIndex = 0;
                Pool autoPool = new Pool();
                foreach (FinalsLink fl in d.FinalsLinks)
                {
                    //each finalslink links a pool placement to a final stage, so if the
finalsindex is not the same as it was for the last final stage, we need a new finalspool
                    if (finalsIndex < fl.Finalstage)
                    {
                        finalsIndex = fl.Finalstage;
                        autoPool = db.PoolSet.Add(new Pool() { Name = d.Name + " " +
(char)(64 + finalsIndex) + " slutspil", Division = d, IsAuto = true });
                    }

                    //foreach pool i the division that is not autogenerated we need one
team representing the poolplacement in each pool
                    foreach (Pool pool in d.Pools)
                    {
                        if (!pool.IsAuto)
                        {
                            //make sure we dont get a team from a pool that does not have
enough teams to have a poolplacement of that value
                            if (pool.Teams.Count >= fl.PoolPlacement)
                            {
                                Team team = db.TeamSet.Add(new Team() { Name = "Nr " +
fl.PoolPlacement + " fra " + d.Name + " - " + pool.Name, PoolPlacement =
fl.PoolPlacement, PrevPool = pool, IsAuto = true, Pool = autoPool });
                                team.TimeIntervals = SameTimeInterval(team.PrevPool);
                            }
                        }
                    }
                }
            }
            db.SaveChanges();
            return true;
        }

        //this function generate the matches for the final stages
        public bool GenerateFinalsMatches(int tournamentID)
        {
            CupDBContainer db = new CupDBContainer();
            Tournament t = db.TournamentSet.Find(tournamentID);
            foreach (Division d in t.Divisions)
            {
                //get all pools that are auto generated
                List<Pool> finalsPools = d.Pools.Where(x => x.IsAuto).ToList();
                foreach (Pool finalPool in finalsPools)
                {
                    if (finalPool.Teams.Count <2)
                    {
                        throw new Exception("not enough teams");
                    }
                    List<Team> teams = new List<Team>();
                    teams.AddRange(finalPool.Teams);


                    //if finals are round robin same as above
                    if (d.TournamentStructure == TournamentStructure.RoundRobin)
                    {
```

```
                        TournamentStage tStage = new TournamentStage();
                        tStage = db.TournamentStageSet.Add(new TournamentStage() { Pool =
finalPool, DivisionTournament = d.DivisionTournament, TournamentStructure =
d.TournamentStructure, TimeInterval = new TimeInterval() { StartTime = DateTime.MinValue,
EndTime = t.TimeIntervals.Last().EndTime } });

                        for (int k = 0; k < teams.Count; k++)
                        {
                            for (int l = k + 1; l < teams.Count; l++)
                            {
                                db.MatchSet.Add(new Match() { Teams = { teams[k],
teams[l] }, TournamentStage = tStage, Duration = d.MatchDuration, Number = matchNumber++
});
                            }
                        }
                    }
                    //if finals are knockout
                    else
                    {
                        Pool KOPool = new Pool();
                        TournamentStage tournyStage = new TournamentStage();
                        int pow = 0;
                        //counts up so 2^pow is as close to but not below the number of
teams
                        while (Math.Pow(2, pow) <= teams.Count)
                        {
                            pow++;
                        }
                        // gets the number of teams that fit into a normal knockout stage
                        int powOfTwo = (int)Math.Pow(2, pow - 1);
                        // if this number is lower than the actual number of teams
                        if (powOfTwo < teams.Count)
                        {
                            //number of teams that need to compete an extra round to
qualify
                            int numOfExtraTeams = (teams.Count - powOfTwo) * 2;
                            //order the list after poolplacement and take the worst
seeded teams from the list and into extraTeams
                            teams = teams.OrderByDescending(x =>
x.PoolPlacement).ToList();
                            List<Team> extraTeams = new List<Team>();
                            extraTeams.AddRange(teams.Take(numOfExtraTeams));
                            teams = teams.Skip(numOfExtraTeams).ToList();
                            //determine what kind of extra round is needed to get the
number of teams down to 2^pow-1
                            switch (powOfTwo)
                            {
                                case 2:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " semi finaler", Division = d, IsAuto = true });
                                    break;
                                case 4:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " kvart finaler", Division = d, IsAuto = true });
                                    break;
                                default:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " " + powOfTwo + ". dels finaler", Division = d, IsAuto = true });
```

```
                                break;
                            }
                            tournyStage = db.TournamentStageSet.Add(new TournamentStage()
{ Pool = KOPool, DivisionTournament = d.DivisionTournament, TournamentStructure =
d.TournamentStructure, TimeInterval = new TimeInterval() { StartTime = DateTime.MinValue,
EndTime = t.TimeIntervals.Last().EndTime } });

                            //matches the first teams agains the last to get the worst
seeds to play agains the best
                            // each match generated also generates a team representing
the winner and adds it back to the original teams list
                            for (int i = 0; i < extraTeams.Count; i++)
                            {

                                if (extraTeams[i].Matches.Count == 0)
                                {
                                    for (int j = extraTeams.Count - 1; j > i; j--)
                                    {
                                        //makes sure the teams are not from the same pool
                                        if (extraTeams[i].PrevPool !=
extraTeams[j].PrevPool && extraTeams[j].Matches.Count == 0)
                                        {
                                            Team winnerTeam = new Team() { Name = "Vinder
af kamp " + matchNumber, IsAuto = true, PrevPool = KOPool };
                                            teams.Add(winnerTeam);
                                            Match m = db.MatchSet.Add(new Match() { Teams
= { extraTeams[i], extraTeams[j] }, Duration = d.MatchDuration, TournamentStage =
tournyStage, Number = matchNumber++ });
                                            break;
                                        }
                                    }
                                    // if the team didnt get set up for a match, every
other team that does not already have a match must be from the same pool
                                    // therefor we just match the best and worst team
against each other

                                    if (extraTeams[i].Matches.Count == 0)
                                    {
                                        for (int j = extraTeams.Count - 1; j > i; j--)
                                        {
                                            if (extraTeams[j].Matches.Count == 0)
                                            {
                                                Team winnerTeam = new Team() { Name =
"Vinder af kamp " + matchNumber, IsAuto = true, PrevPool = KOPool };
                                                teams.Add(winnerTeam);
                                                Match m = db.MatchSet.Add(new Match() {
Teams = { extraTeams[i], extraTeams[j] }, Duration = d.MatchDuration, TournamentStage =
tournyStage, Number = matchNumber++ });
                                                break;
                                            }
                                        }
                                    }
                                }
                                extraTeams[i].Pool = KOPool;
                                db.TeamSet.Add(extraTeams[i]);
                                extraTeams[i].TimeIntervals =
SameTimeInterval(extraTeams[i].PrevPool);
                            }
```

```
                        }
                        List<Team> teamsToAdd = new List<Team>();
                        //this while loop keeps going untill there is no more teams
                        // each round all the teams are taken from the list, which makes
it empty
                        // but the winners of all matches generated this iteration gets
added back into the team list
                        // this continues untill only two teams are left and no more
teams are added back into the team list
                        while (teams.Count > 0)
                        {
                            teamsToAdd.AddRange(teams);
                            teams.Clear();
                            //determines the kind of round we have gotten to at this
point
                            switch (teamsToAdd.Count)
                            {
                                case 1:
                                    throw new Exception("Not enough teams");
                                case 2:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " finale", Division = d, IsAuto = true });
                                    break;
                                case 4:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " semi finaler", Division = d, IsAuto = true });
                                    break;
                                case 8:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " kvart finaler", Division = d, IsAuto = true });
                                    break;
                                default:
                                    KOPool = db.PoolSet.Add(new Pool() { Name =
finalPool.Name + " " + teamsToAdd.Count / 2 + ". dels finaler", Division = d, IsAuto =
true });
                                    break;
                            }

                            if (teams.Count != 1)
                            {
                                tournyStage = db.TournamentStageSet.Add(new
TournamentStage() { Pool = KOPool, DivisionTournament = d.DivisionTournament,
TournamentStructure = d.TournamentStructure, TimeInterval = new TimeInterval() {
StartTime = DateTime.MinValue, EndTime = t.TimeIntervals.Last().EndTime } });
                                // if there is 2 teams we are at the finals and no winner
team should be added to the list, we are done
                                if (teamsToAdd.Count == 2)
                                {
                                    Match m = db.MatchSet.Add(new Match() { Teams = {
teamsToAdd[0], teamsToAdd[1] }, Duration = d.MatchDuration, TournamentStage =
tournyStage, Number = matchNumber++ });
                                    teamsToAdd[0].Pool = KOPool;
                                    teamsToAdd[1].Pool = KOPool;
                                    db.TeamSet.AddRange(teamsToAdd);
                                    teamsToAdd[0].TimeIntervals =
SameTimeInterval(teamsToAdd[0].PrevPool);
                                    teamsToAdd[1].TimeIntervals =
SameTimeInterval(teamsToAdd[1].PrevPool);
```

```
                                }
                                else
                                {
                                    //same procedure as above
                                    for (int i = 0; i < teamsToAdd.Count; i++)
                                    {

                                        if (teamsToAdd[i].Matches.Count == 0)
                                        {
                                            for (int j = teamsToAdd.Count - 1; j > i; j--
)
                                            {
                                                if (teamsToAdd[i].PrevPool !=
teamsToAdd[j].PrevPool && teamsToAdd[j].Matches.Count == 0)
                                                {
                                                    Team winnerTeam = new Team() { Name =
"Vinder af kamp " + matchNumber, IsAuto = true, PrevPool = KOPool };
                                                    teams.Add(winnerTeam);
                                                    Match m = db.MatchSet.Add(new Match()
{ Teams = { teamsToAdd[i], teamsToAdd[j] }, Duration = d.MatchDuration, TournamentStage =
tournyStage, Number = matchNumber++ });
                                                    break;
                                                }
                                            }
                                            if (teamsToAdd[i].Matches.Count == 0)
                                            {
                                                for (int j = teamsToAdd.Count - 1; j > i;
j--)
                                                {
                                                    if (teamsToAdd[j].Matches.Count == 0)
                                                    {
                                                        Team winnerTeam = new Team() {
Name = "Vinder af kamp " + matchNumber, IsAuto = true, PrevPool = KOPool };
                                                        teams.Add(winnerTeam);
                                                        Match m = db.MatchSet.Add(new
Match() { Teams = { teamsToAdd[i], teamsToAdd[j] }, Duration = d.MatchDuration,
TournamentStage = tournyStage, Number = matchNumber++ });
                                                        break;
                                                    }
                                                }
                                            }
                                        }
                                        teamsToAdd[i].Pool = KOPool;
                                        db.TeamSet.Add(teamsToAdd[i]);
                                        teamsToAdd[i].TimeIntervals =
SameTimeInterval(teamsToAdd[i].PrevPool);
                                    }
                                }
                            }
                            teamsToAdd.Clear();
                            db.SaveChanges();
                        }
                        List<Team> teamsToClearUp = db.TeamSet.Where(x => x.Pool.Id ==
finalPool.Id).ToList();
                        foreach (Team team in teamsToClearUp)
                        {
                            db.TimeIntervalSet.RemoveRange(team.TimeIntervals);
                        }
```

```
                        db.TeamSet.RemoveRange(teamsToClearUp);
                        db.PoolSet.Remove(finalPool);
                        db.SaveChanges();
                    }

                }
            }
            db.SaveChanges();
            return true;
        }


        // Returns a list of TimeInterval, where each DateTime in a TimeInterval is equal
to the highest time of the day (for the start of a day)
        // or lowest time of the day (for the end of the day) in the teams in a pool.
        public List<TimeInterval> SameTimeInterval(Pool p)
        {
            List<TimeInterval> intervals = new List<TimeInterval>();
            for (int i = 0; i < p.Division.Tournament.TimeIntervals.Count; i++)
            {
                DateTime dtStart = p.Teams.Select(x =>
x.TimeIntervals.ToArray()[i].StartTime).OrderByDescending(x => x.TimeOfDay).First();
                DateTime dtEnd = p.Teams.Select(x =>
x.TimeIntervals.ToArray()[i].EndTime).OrderBy(x => x.TimeOfDay).First();
                intervals.Add(new TimeInterval() { StartTime = dtStart, EndTime = dtEnd
});
            }
            return intervals;
        }


    }
}
```