```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Data.Entity;
using CupPlaner.Helpers;

namespace CupPlaner.Controllers
{
    // Field controller with CRUD functions and a function to get all the fields in a
tournament
    public class FieldController : Controller
    {
        // Database container, has functionalities to connect to the database classes.
        CupDBContainer db = new CupDBContainer();
        ScheduleManager sm = new ScheduleManager();

        // GET: Field/Details/5 - Fetches the details of the class, takes the "id"
parameter to determine the corresponding Field object.
        // Returns a Json object, which contains a copy of the corresponding Field
variables.
        public ActionResult Details(int id)
        {
            try
            {
                Field f = db.FieldSet.Find(id);
                object obj = new { status = "success", Id = f.Id, name = f.Name, size =
f.Size };
                return Json(obj, JsonRequestBehavior.AllowGet);
            }
            catch (Exception ex)
            {
                return Json(new { status = "error", message = "Could not find field",
details = ex.Message }, JsonRequestBehavior.AllowGet);
            }
        }
        // The function (GetAllTournament) tries to find a tournament in the database,
        // corresponding to the given tournamentId. Copies the tournaments lists of
        // fields, matches and timeintervals, and returns a JSON object
        // indicating whether the action was successful or not, finding the fields.
        public ActionResult GetAllTournamentFields(int tournamentId)
        {
            try
            {
                Tournament t = db.TournamentSet.Find(tournamentId);
                List<object> fields = new List<object>();
                List<object> matches = new List<object>();
                List<object> timeintervals = new List<object>();

                // Get all fields and matches for each one
                if(t.Fields != null)
                {
                    foreach(Field f in t.Fields)
                    {
                        //if (f.Matches.Count > 0)
                        //{
                            matches = new List<object>();
                            foreach (Match m in f.Matches.OrderBy(match =>
match.StartTime))
                            {
                                matches.Add(new { Id = m.Id, StartTime = m.StartTime,
EndTime = m.StartTime.AddMinutes(m.Duration), Duration = m.Duration, Date =
m.StartTime.Date, Nr = m.Number, Pool = m.TournamentStage.Pool.Name, HomeTeam = new { Id
= m.Teams.First().Id, Name = m.Teams.First().Name }, AwayTeam = new { Id =
m.Teams.Last().Id, Name = m.Teams.Last().Name } });
                            }
                            fields.Add(new { Id = f.Id, Name = f.Name, fieldSize =
f.Size, matches = matches });
                        //}
                    }
                    foreach (TimeInterval ti in t.TimeIntervals)
                    {
                        List<int> times = new List<int>();

                        int latest = ti.StartTime.Hour;
                        int timeIndex = 0;
                        while (latest < ti.EndTime.Hour)
                        {
                            latest = ti.StartTime.Hour + timeIndex;
                            times.Add(latest);
                            timeIndex++;
                        }
                        timeintervals.Add(new { StartTime = ti.StartTime, EndTime =
ti.EndTime, Date = ti.StartTime.Date, Times = times });
                    }
                }
                object obj = new { status = "success", Fields = fields, TimeIntervals =
timeintervals };
                return Json(obj, JsonRequestBehavior.AllowGet);
            }
            catch(Exception ex)
            {
                return Json(new { status = "error", message = "Could not find fields",
details = ex.Message }, JsonRequestBehavior.AllowGet);
            }
        }

        // POST: Field/Create - Tries to create a Field object, with the parameters
"name" and "size".
        // Sets the FieldSize and Name to the parameters ("name" and "size").
        // Adds the Field object to the database FieldSet, and saves the changes in the
database.
        // Returns a Json object with a state, indicating whether it succeeded creating
the Field object or not.
        [HttpPost]
        public ActionResult Create(string name, int size, int tournamentId)
        {
            try
            {
                Tournament t = db.TournamentSet.Find(tournamentId);
                Field f = db.FieldSet.Add(new Field() { Name = name, Size =
(FieldSize)size, Tournament = t });
                // Set the next free time of the field for each day to that of the
tournament
                foreach (TimeInterval ti in t.TimeIntervals)
                {
                    f.NextFreeTime.Add(new NextFreeTime() { FreeTime = ti.StartTime });
                }

                db.SaveChanges();

                return Json(new { status = "success", message = "New field added", id =
f.Id, fieldName = f.Name }, JsonRequestBehavior.AllowGet);
            }
            catch (Exception ex)
            {
                return Json(new { status = "error", message = "New field not added",
details = ex.Message }, JsonRequestBehavior.AllowGet);
            }
        }

        // POST: Field/Delete/5 - Tries to delete a Field object, determined by the "id".
        // Deletes the Field object from the FieldSet in the database, and saves the
changes, if succeeded.
        // Returns a Json object, indicating whether it succeeded deleting the Field or
not.
        [HttpPost]
        public ActionResult Delete(int id)
        {
            try
            {
                Field f = db.FieldSet.Find(id);
                Tournament t = db.TournamentSet.Find(f.Tournament.Id);
                // Clear the schedule
                sm.DeleteSchedule(t.Id, db);

                // Remove dependencies
                foreach (Division d in t.Divisions)
                {
                    foreach(Pool p in d.Pools)
                    {
                        foreach(Field favField in p.FavoriteFields.ToList())
                        {

                            if(favField.Id == f.Id)
                            {
                                p.FavoriteFields.Remove(favField);
                            }
                        }
                    }
                }
                db.NextFreeTimeSet.RemoveRange(f.NextFreeTime);
                db.FieldSet.Remove(f);
                db.SaveChanges();

                return Json(new { status = "success", message = "Field deleted" });
            }
            catch (Exception ex)
            {
                return Json(new { status = "error", message = "Field not deleted",
details = ex.Message });
            }
        }
        }
}
```