ScoreApi Summary

Score.cs

```
public class Score {
  public int Id {
    get;
    set;
  }

  [Required, MinLength(3), MaxLength(100)]
  public string Name {
    get;
    set;
  } = string.Empty;

  [Required, MinLength(3), MaxLength(500)]
  public string Module {
    get;
    set;
  } = string.Empty;

  [Required, Column(TypeName = "decimal(5,2)")]
  public double Mark {
    get;
    set;
  } = 0;

  [Column(TypeName = "datetime")]
  public DateTime CreatedAt {
    get;
    set;
  }
}
```

Notes

- `Required`: Not Null
- `decimal(5,2)`: Five digits with two digits being the decimal place.
- `Id`: In this case, is by default, the **auto-incremented primary key.**

ScoreController.cs

```
[ApiController]
[Route("score/[action]")]
public class ScoreController(MyDbContext context): ControllerBase {
  private readonly MyDbContext _context = context;

  [HttpGet]
  public IActionResult GetAll() {
    IQueryable < Score > result = _context.Scores
      .OrderByDescending(x => x.CreatedAt);
    return Ok(result);
  }

  [HttpPost]
  public IActionResult AddScore(Score score) {
    Score scoreSchema = new Score() {
      Name = score.Name,
        Module = score.Module,
        Mark = score.Mark,
        CreatedAt = DateTime.Now
    };

    if (score.Mark < 0 || score.Mark > 100) {
      return BadRequest("Score must be between 0 and 100");
    }

    bool hasDuplicateScore = _context.Scores
      .Any(score1 => score1.Name == score.Name &&
        score1.Module == score.Module &&
        score1.Mark == score.Mark
      );

    if (hasDuplicateScore) {
      return BadRequest("Already exists!");
    }

    _context.Scores.Add(scoreSchema);
    _context.SaveChanges();
    return Ok(scoreSchema);
  }

  [HttpGet]
  [Route("{id}")]
  public IActionResult GetScoreById(int id) {
    Score ? scoreSchema = _context.Scores.Find(id);
    if (scoreSchema == null) {
      return NotFound("Score with id " + id + " not found");
    }
```

```
  return Ok(scoreSchema);
}

[HttpDelete]
[Route("{id}")]
public IActionResult DeleteScoreById(int id) {
  Score ? scoreSchema = _context.Scores.Find(id);
  if (scoreSchema == null) {
    return NotFound("Score with id " + id + " not found");
  }

  _context.Scores.Remove(scoreSchema);
  _context.SaveChanges();
  return Ok(scoreSchema.Id + " has been removed!");
}

[HttpPut]
[Route("{id}")]
public IActionResult UpdateScoreById(int id, Score score) {
  Score ? scoreSchema = _context.Scores.Find(id);
  if (scoreSchema == null) {
    return NotFound("Score with id " + id + " not found");
  }

  scoreSchema.Name = score.Name;
  scoreSchema.Mark = score.Mark;
  scoreSchema.Module = score.Module;

  _context.SaveChanges();
  return NoContent();
}

[HttpGet]
[Route("{name}")]
public IActionResult GetScoreByName(string name) {
  IQueryable < Score > result = _context.Scores
    .Where(scoreSchema => scoreSchema.Name.ToLower().Equals(name.ToLower())
  return Ok(result);
}

[HttpGet]
[Route("{module}")]
public IActionResult GetGradeAScoreByModule(string module) {
  if (string.IsNullOrWhiteSpace(module)) {
    return BadRequest("Module is required!");
  }
```

```
    var result = _context.Scores
      .Where(score => score.Module.Equals(module))
      .Where(score => score.Mark >= 80)
      .Select(score => new {
        score.Name, score.Mark
      })
      .OrderByDescending(score => score.Mark);

    if (result.Count() == 0) {
      return NotFound("No such a module - " + module + " !");
    }

    return Ok(result);
  }

  [HttpGet]
  public IActionResult GetAverageGradeOfEachModule() {
    var result = _context.Scores
      .GroupBy(score => score.Module)
      .Select(group => new {
        Module = group.Key, Mark = group.Average(score => score.Mark)
      })
      .OrderByDescending(score => score.Mark)
      .ToList();

    return Ok(result);
  }
}
```

Notes

- `_context.Scores`: Returns a list of `Score` in the existing database. e.g. `[Score@1, Score@2, ..., Score@n]`.
- Methods that can be used for chain operation:
  - `Where`: filters out those items that do not satisfy the condition.
    - Example of getting scores whose mark are at least 80.
      `scores.Where(score => score.Mark >= 80)`
  - `OrderByDescending`: orders the list based on an attribute.
    - Example of ordering scores my mark in descending order.
      `scores.OrderByDescending(score => score.Mark)`
  - `Select`: Selects only particular fields to return.
    - Example of selecting only Name and Mark of scores.
      `scores.Select(score => new { score.Name, score.Mark })`

- Other methods:
  - `Find`: Finds an item by primary key.
    - Example of finding the item with primary key `id` being `2`.
      `scores.Find(2)`
  - `Any`: Returns `true` if any of the item meets the condition.
    - Example of checking if there is any score with mark being at least 90.
      `scores.Any(score => score.Mark >= 90)`
  - `All`: Returns `true` if all the items meet the condition.
    - Example of checking if all scores have mark at least 90.
      `scores.All(score => score.Mark >= 90)`
  - `Count`: Returns the length of the list.

Example of some complicated queries

1. Get all scores that belong to the modules `edp` and `mad`, omit those scores whose mark is below `50`, sort by `mark`.

```
scores
    .Where(score => score.Module.Equals("edp") || score.Module.Equals("mad
    .Where(score => score.Mark >= 50)
    .OrderByDescending(score => score.Mark);
```

2. Check if there is two or more scores from `edp` with the same mark.

```
scores
    .Where(score => score.Module.Equals("edp"))
    .Any(score1 => scores.Any(score2 => score1.Id != score2.Id
                            && score1.Mark == score2.Mark));
```

3. Returns the name of those students who got their marks above `90` for **all modules** he had taken. *(Or in other words, there does not exist any module which the student got below 90)*

```
// Approach 1: The student scores 90 and above for all modules
scores
    .Where(score1 =>
            scores.Where(score2 => score1.Name.Equals(score2.Name))
                        .All(score2 => score2.Mark >= 90))
    .Select(score => score.Name)
    .Distinct();

// Approcah 2: Not any module that the student scores below 90
scores
    .Where(score1 =>
            !scores.Where(score2 => score1.Name.Equals(score2.Name))
```

```
                        .Any(score2 => score2.Mark < 90))
    .Select(score => score.Name)
    .Distinct();
```