Lecture 04 - API

1. Open Visual Studio Code.
2. Open folder which would contain your project. File —> Open Folder. e.g. *Lecture04/CourseAdminSystem/Backend*.
3. Open Terminal from menu.
4. Create a new solution using the following command

```
dotnet new sln --name CourseAdminSystem
```

5. Add a new API project within the current folder using the following command.

```
dotnet new webapi -n CourseAdminSystem.API --use-controllers
```

6. Add the newly created project to the existing solution using the following command.

```
dotnet sln add CourseAdminSystem.API
```

7. Open the *Program.cs* file and remove the following line.

```
app.UseHttpsRedirection();
```

7. Run the API to see if everything works.

Create database using the script provided in Canvas. Please use PgAdmin Tool to create the database.

Add a new Model project to the solution

The idea is to have a new project to handle all the code responsible for database management.

1. Add a new *ClassLibrary* project to the existing solution using the following command. Please make sure that your terminal is pointing at the correct folder, which is Backend in this case.

```
dotnet new classlib -n CourseAdminSystem.Model
```

2. Add the newly created project to the existing solution using the following command.

```
dotnet sln add CourseAdminSystem.Model
```

3. In order to be able to “talk” to our Postgres database and for handling configuration, we’ll have to add a few nuget packages to our project using the following command.

```
dotnet add CourseAdminSystem.Model package NpgSql
dotnet add CourseAdminSystem.Model package Microsoft.Extensions.Configuration
```

Prepare Model project with Repositories

The idea is to have (ideally) one repository each for each of the tables in the database.

1. Go to the appsettings.json file in the API project, and all the following json configuring the connection string to the database. Please note that you might have to make minor changes to it e.g. based on the name of the database and/or credentials.

```
...
"ConnectionStrings": {
  "AppProgDb": "Host=localhost:5432;Username=postgres;Password=;Database=AppProgDb"
}
...
```

1. Add two new folders in the Model project. The first one is *Entities* and the second one is *Repositories*.
2. For first time only, create a class *BaseRepository* with the following code. This class will ensure that all your repositories have the necessary code to read/write to the database.

```
public class BaseRepository
{
   protected string ConnectionString {get;}
   public BaseRepository(IConfiguration configuration) {
      ConnectionString = configuration.GetConnectionString("AppProgDb");
   }

   protected NpgsqlDataReader GetData(NpgsqlConnection conn, NpgsqlCommand cmd)
   {
      conn.Open();
      return cmd.ExecuteReader();
   }

   protected bool InsertData(NpgsqlConnection conn, NpgsqlCommand cmd)
   {
      conn.Open();
      cmd.ExecuteNonQuery();
      return true;
   }

   protected bool UpdateData(NpgsqlConnection conn, NpgsqlCommand cmd)
   {
      conn.Open();
      cmd.ExecuteNonQuery();
      return true;
   }

   protected bool DeleteData(NpgsqlConnection conn, NpgsqlCommand cmd)
   {
      conn.Open();
      cmd.ExecuteNonQuery();
      return true;
   }
}
```

For each new Repository, follow the following instructions. The following example is based on Student.

1. Under the *Entities* folder, create a new class called *Student* with the following code.

```
public class Student {
   public Student(int id){Id = id;}
   public int Id { get; set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
   public int StudyProgramId { get; set; }
   public DateTime DateOfBirth { get; set; }
   public string Email { get; set; }
   public string Phone { get; set; }
}
```

3. Under the *Repositories* folder, create a new class called *StudentRepository*. Remember to inherit it from *BaseRepository* to make sure that all your repositories have the necessary code to read/write to the database.
4. Add reference of the Model project into API project using the following command in Terminal. Make sure you are at the solution level.

```
dotnet add CourseAdminSystem.API/CourseAdminSystem.API.csproj reference
CourseAdminSystem.Model/CourseAdminSystem.Model.csproj
```

5. For each repository created add the following line in program file to make the respository available to API Controller through dependency injection.

```
builder.Services.AddScoped<StudentRepository, StudentRepository>();
```

6. Within each repository we have to add the code for C(reate), R(emove), U(pdate), D(elete) operations. An example for the related methods is as below.

```
using System;
using CourseAdminSystem.Model.Entities;
using Microsoft.Extensions.Configuration;
using Npgsql;
using NpgsqlTypes;

namespace CourseAdminSystem.Model.Repositories;

public class StudentRepository : BaseRepository
{
   public StudentRepository(IConfiguration configuration) : base(configuration)
   {
   }
```

```
public Student GetStudentById(int id)
{
   NpgsqlConnection dbConn = null;
   try
   {
      //create a new connection for database
      dbConn = new NpgsqlConnection(ConnectionString);

      //creating an SQL command
      var cmd = dbConn.CreateCommand();
      cmd.CommandText = "select * from student where id = @id";

      cmd.Parameters.Add("@id", NpgsqlDbType.Integer).Value = id;

      //call the base method to get data
      var data = GetData(dbConn, cmd);

      if (data != null)
      {
         if (data.Read()) //every time loop runs it reads next like from fetched rows
         {
            return new Student(Convert.ToInt32(data["id"]))
            {
               FirstName = data["firstname"].ToString(),
               LastName = data["lastname"].ToString(),
               StudyProgramId = (int)data["studyprogramid"],
               DateOfBirth = Convert.ToDateTime(data["dob"]),
               Email = data["email"].ToString(),
               Phone = data["phone"].ToString()
            };
         }
      }

      return null;
   }
   finally
   {
      dbConn?.Close();
   }
}
```

```
public List<Student> GetStudents()
{
   NpgsqlConnection dbConn = null;
   var students = new List<Student>();
   try
   {
      //create a new connection for database
      dbConn = new NpgsqlConnection(ConnectionString);

      //creating an SQL command
      var cmd = dbConn.CreateCommand();
      cmd.CommandText = "select * from student";

      //call the base method to get data
      var data = GetData(dbConn, cmd);

      if (data != null)
      {
         while (data.Read()) //every time loop runs it reads next like from fetched rows
         {
            Student s = new Student(Convert.ToInt32(data["id"]))
            {
               FirstName = data["firstname"].ToString(),
               LastName = data["lastname"].ToString(),
               StudyProgramId = (int)data["studyprogramid"],
               DateOfBirth = Convert.ToDateTime(data["dob"]),
               Email = data["email"].ToString(),
               Phone = data["phone"].ToString()
            };

            students.Add(s);
         }
      }

      return students;
   }
   finally
   {
      dbConn?.Close();
   }
}
```

```
   //add a new student
   public bool InsertStudent(Student s)
   {
      NpgsqlConnection dbConn = null;
      try
      {
         dbConn = new NpgsqlConnection(ConnectionString);
         var cmd = dbConn.CreateCommand();
         cmd.CommandText = @"
insert into student
(firstname,lastname, studyprogramid, dob, email, phone)
values
(@firstname,@lastname, @studyprogramid, @dob, @email, @phone)
";

         //adding parameters in a better way
         cmd.Parameters.AddWithValue("@firstname", NpgsqlDbType.Text, s.FirstName);
         cmd.Parameters.AddWithValue("@lastname", NpgsqlDbType.Text, s.LastName);
         cmd.Parameters.AddWithValue("@studyprogramid", NpgsqlDbType.Integer,
s.StudyProgramId);
         cmd.Parameters.AddWithValue("@dob", NpgsqlDbType.Date, s.DateOfBirth);
         cmd.Parameters.AddWithValue("@email", NpgsqlDbType.Text, s.Email);
         cmd.Parameters.AddWithValue("@phone", NpgsqlDbType.Text, s.Phone);

         //will return true if all goes well
         bool result = InsertData(dbConn, cmd);

         return result;
      }
      finally
      {
         dbConn?.Close();
      }
   }

   public bool UpdateStudent(Student s)
   {
      var dbConn = new NpgsqlConnection(ConnectionString);
      var cmd = dbConn.CreateCommand();
      cmd.CommandText = @"
update student set
    firstname=@firstname,
    lastname=@lastname,
    studyprogramid=@studyprogramid,
    dob=@dob,
    email=@email,
    phone=@phone
where
id = @id";

      cmd.Parameters.AddWithValue("@firstname", NpgsqlDbType.Text, s.FirstName);
      cmd.Parameters.AddWithValue("@lastname", NpgsqlDbType.Text, s.LastName);
      cmd.Parameters.AddWithValue("@studyprogramid", NpgsqlDbType.Integer,
s.StudyProgramId);
      cmd.Parameters.AddWithValue("@dob", NpgsqlDbType.Date, s.DateOfBirth);
      cmd.Parameters.AddWithValue("@email", NpgsqlDbType.Text, s.Email);
      cmd.Parameters.AddWithValue("@phone", NpgsqlDbType.Text, s.Phone);
      cmd.Parameters.AddWithValue("@id", NpgsqlDbType.Integer, s.Id);

      bool result = UpdateData(dbConn, cmd);
      return result;
   }

   public bool DeleteStudent(int id)
   {
      var dbConn = new NpgsqlConnection(ConnectionString);
      var cmd = dbConn.CreateCommand();
      cmd.CommandText = @"
delete from student
where id = @id
";

      //adding parameters in a better way
      cmd.Parameters.AddWithValue("@id", NpgsqlDbType.Integer, id);

      //will return true if all goes well
      bool result = DeleteData(dbConn, cmd);

      return result;
   }
}
```

Add a new Controller to the API project

The followimg steps need to be done to add each new controller to the project. The following example displayed the controller or the *Student*.

1. Right click on the Controllers folder in the API project and select *New file* and then select *API Controller*.
2. Then copy the following code in the file. Please note that you might have to make some changes to the namespaces if your directory structure is a bit different.

```
using CourseAdminSystem.Model.Entities;
using CourseAdminSystem.Model.Repositories;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CourseAdminSystem.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class StudentController : ControllerBase
    {
        protected StudentRepository Repository {get;}

        public StudentController(StudentRepository repository) {
            Repository = repository;
        }

        [HttpGet("{id}")]
        public ActionResult<Student> GetStudent([FromRoute] int id)
        {
            Student student = Repository.GetStudentById(id);
            if (student == null) {
                return NotFound();
            }

            return Ok(student);
        }

        [HttpGet]
        public ActionResult<IEnumerable<Student>> GetStudents()
        {
            return Ok(Repository.GetStudents());
        }

        [HttpPost]
        public ActionResult Post([FromBody] Student student) {
            if (student == null)
            {
                return BadRequest("Student info not correct");
            }

            bool status = Repository.InsertStudent(student);
            if (status)
            {
                return Ok();
            }

            return BadRequest();
        }

        [HttpPut]
        public ActionResult UpdateStudent([FromBody] Student student)
        {
            if (student == null)
            {
                return BadRequest("Student info not correct");
            }

            Student existinStudent = Repository.GetStudentById(student.Id);
            if (existinStudent == null)
            {
                return NotFound($"Student with id {student.Id} not found");
            }

            bool status = Repository.UpdateStudent(student);
            if (status)
            {
                return Ok();
            }

            return BadRequest("Something went wrong");
        }

        [HttpDelete("{id}")]
        public ActionResult DeleteStudent([FromRoute] int id) {
            Student existingStudent = Repository.GetStudentById(id);
            if (existingStudent == null)
            {
                return NotFound($"Student with id {id} not found");
            }

            bool status = Repository.DeleteStudent(id);
            if (status)
            {
                return NoContent();
            }

            return BadRequest($"Unable to delete student with id {id}");
        }
    }
}
```

MISC

1. To view the available templates, run the following command.

```
dotnet new --list
```

2. To view solution explorer, CTRL + ALT + L, or right click on the folder and then enable Solution Explorer.