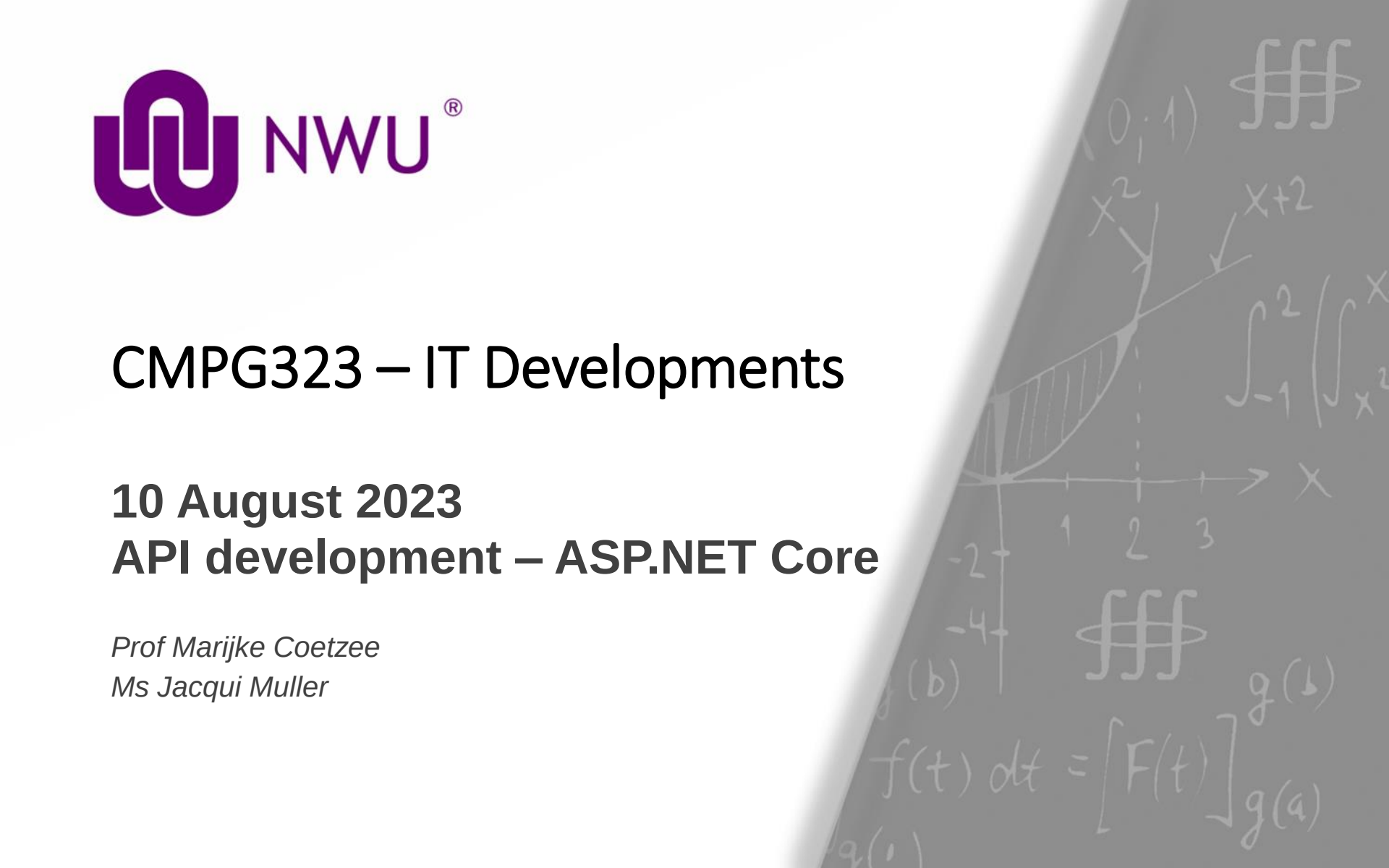


CMPG323 – IT Developments

10 August 2023
API development – ASP.NET Core

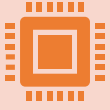
Prof Marijke Coetzee
Ms Jacqui Muller

Overview

1. Controllers
2. The Entity Framework Core
3. Demo

- https://survey.stackoverflow.co/2023/

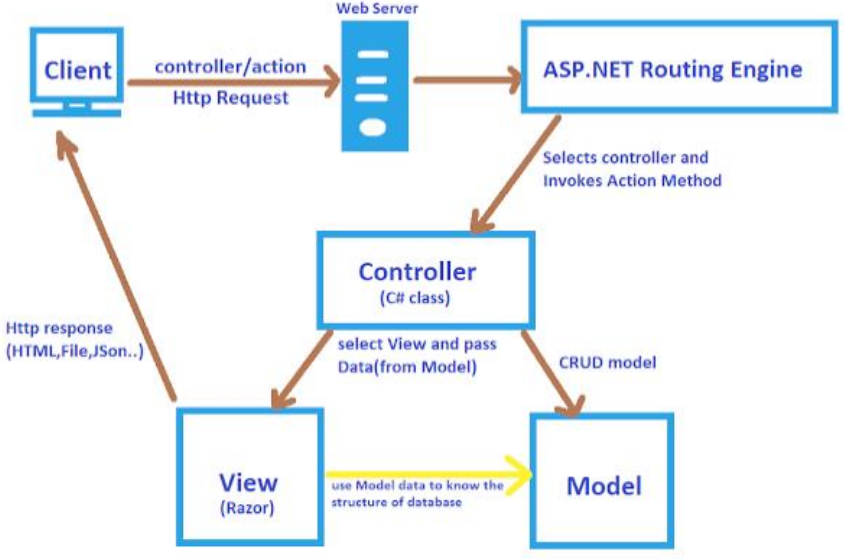
ASP.NET Web API Controllers



Model View Controller (MVC) is a software design pattern for implementing web applications with user interfaces, and it is used to separate the concerns of three major components, which are **models, views, and controllers**.

This architectural pattern has been around for many years, and it strives to promote code reuse and simultaneous development.

ASP.NET Web API Controllers



Controllers are the main **entry point** and handle requests initiated from user interaction.

Logic is performed from within the controller, and then it potentially creates a *model*, which houses the **state of the application** and the business logic around it.

The model is then passed by the controller to a *view*, which has the responsibility of **rendering a user interface**, possibly containing the data from the model.

ASP.NET Web API Controllers

Controllers are used to logically group a section of an application with a set of standard actions or **endpoints**.

They provide the infrastructure for executing action methods.

Web API controller is a class that can be created under the **Controllers folder** or any other folder under your project's root folder.

The name of a controller class must end with "Controller" and it must be derived from System.Web.Http.**ApiController** class.

All the public methods of the controller are called **action methods**.

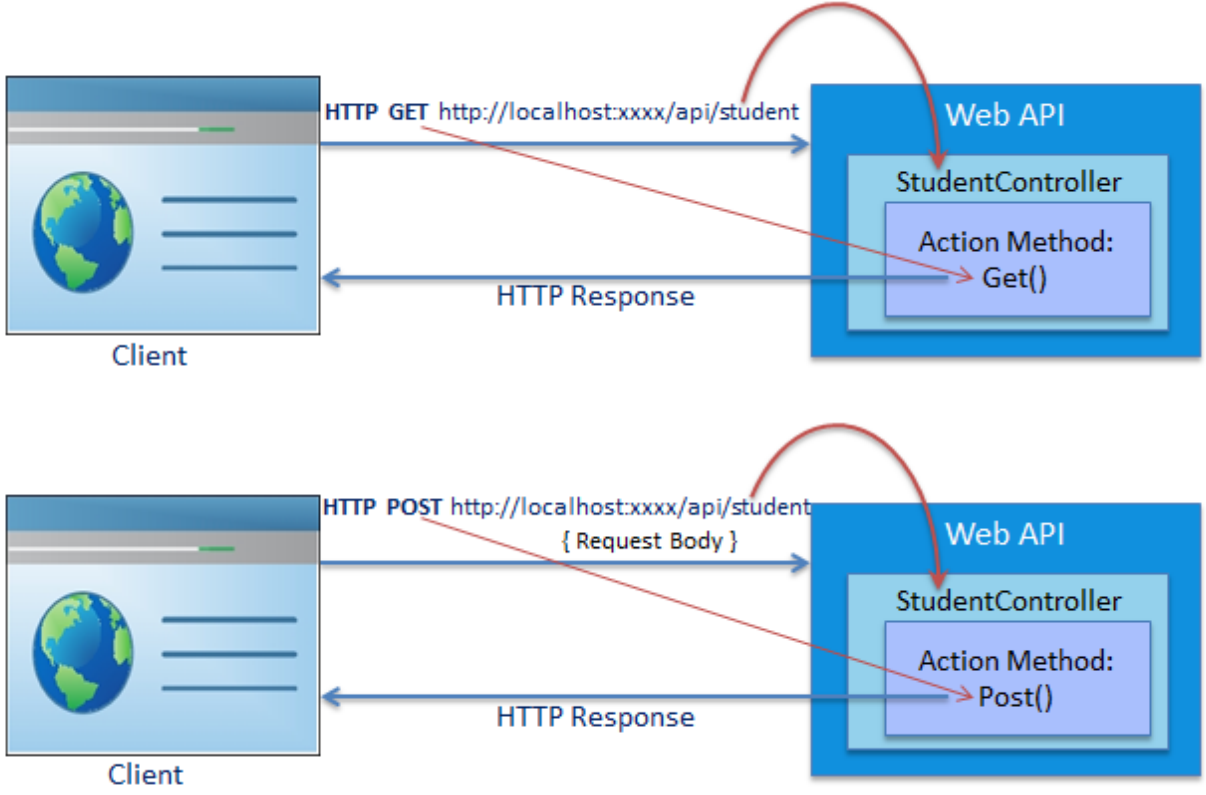
Based on the incoming request URL and HTTP verb (GET/POST/PUT/PATCH/DELETE), Web API decides which Web API controller and action method to execute

ASP.NET Web API Controllers

```
[Route("api/[controller]")]
[ApiController]
StudentsController : ApiController
  {
    // GET: api/student
    public IEnumerable<string> Get()
    {
      return new string[] { "value1", "value2" };
    }
    // GET: api/student/5
    public string Get(int id)
    {
      return "value";
    }
```

```
// POST: api/student
public void Post([FromBody]string value)
{
}

// PUT: api/student/5
public void Put(int id, [FromBody]string value)
{
}

// DELETE: api/student/5
public void Delete(int id)
{
```

ASP.NET Web API Controllers



Web API Routing

Entity Framework Core

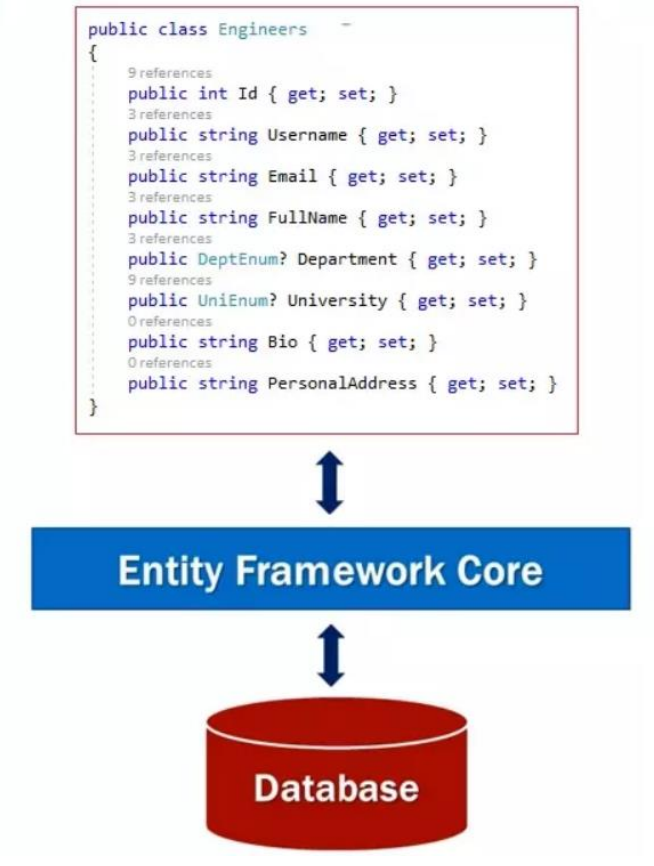
Entity Framework (EF) Core is an **ORM** (Object-Relational Mapper) Framework for data access in .NET Core

ORM automatically **creates classes** based on **database tables** and vice versa is also true.

O/RMs work by mapping between two worlds: the relational database and the object-oriented software world of classes and software code.

EF Core's main strength is allowing software developers to write database access code quickly in a language that they know better than SQL.

Entity Framework Core

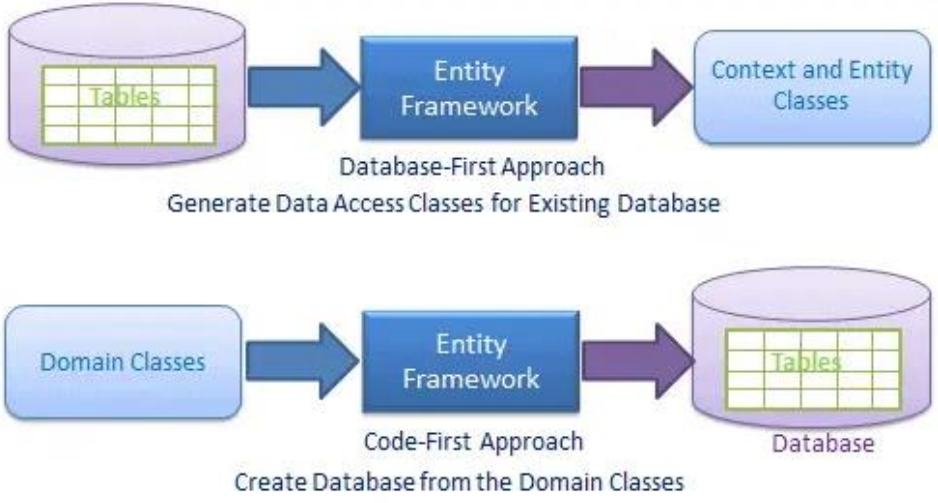


Entity Framework Core

EF Core mapping between a database and .NET software

Relational database	.NET software
Table	.NET class
Table columns	Class properties/fields
Rows	Elements in .NET collections—for instance, List
Primary keys: unique row	A unique class instance
Foreign keys: define a relationship	Reference to another class
SQL—for instance, WHERE	.NET LINQ—for instance, Where(p => ...

Entity Framework Core



We can use two approaches in Entity Framework Core, which are:

- Code First Approach
- Database First Approach

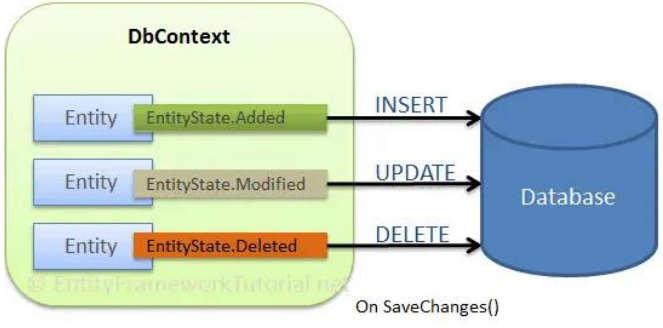
Entity Framework Core

Database First Approach

Create a model for an **existing database** in Entity Framework Core

Reverse engineering is the process of **scaffolding entity type classes** and a **DbContext class** based on a database schema. This reverse engineering command **creates entity and context classes**

An instance of **DbContext** represents a session with the database which can be used to query and save instances of your entities to a database.



Entity Framework Core

The most basic unit in Entity Framework Core is the **model**

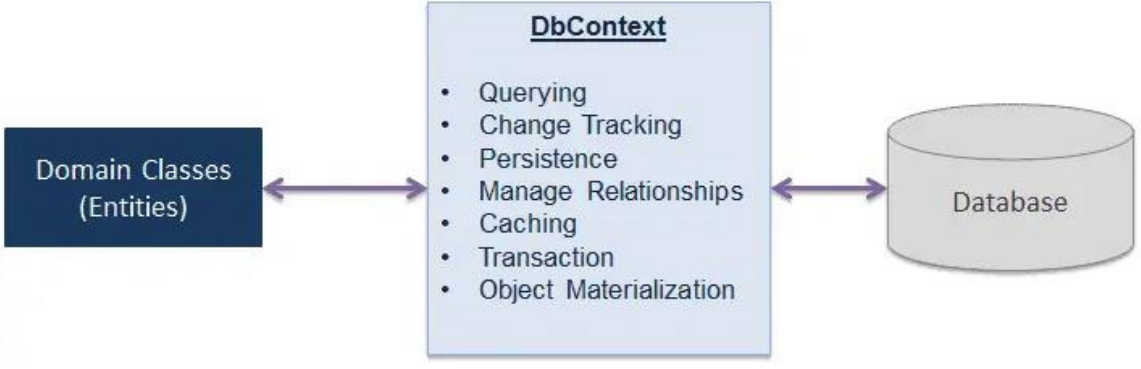
```
public class Employee
{
    public int EmployeeId { get; set; }
    public string EmployeeName { get; set; }
    public int DepartmentId { get; set; }

    public virtual Department Departments { get; set; }
}
```

The **DbSet** class represents an entity set that can be used for create, read, update, and delete operations

```
public class EmployeeManagementContext : DbContext
{
    0 references
    public DbSet<Employee> Employees { get; set; }
    0 references
    public DbSet<Department> Departments { get; set; }
}
```

Entity Framework Core

DbContext is a bridge between your domain or entity classes and the database.



To do this week

Do the module training

Homework exercise 5:

- **Implement the basic API with a local database** as explained in this session. You must complete this step successfully this week, to be able to progress to the next step, where the DB is hosted in the cloud.
- Create controllers for all three databases
- Upload a screenshot of your work

DEMO