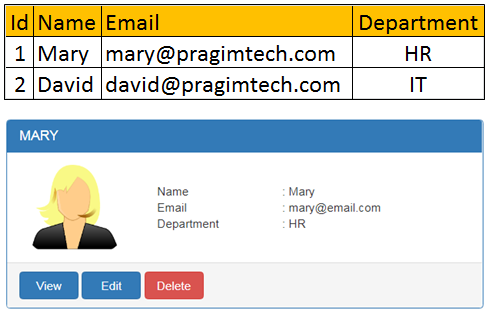
**Model in ASP.NET Core MVC**

In this video we will discuss **Model in ASP.NET Core MVC**with an example.    
  
  
We want to ultimately, retrieve a specific employee details from the **Employees**database table and display on the web page as shown below.   
  
   
  
  
**Model in MVC** contains a set of classes that represent data and the logic to manage that data. So, to represent the employee data that we want to display, we use the following Employee class.

public class Employee  
{  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public string Email { get; set; }  
    public string Department { get; set; }  
}

Model classes in ASP.NET Core does not have to be in the **Models** folder, but keeping them in a folder named **Models**is a good practice as it is easier to find them later.   
  
In addition to the Employee class that represent the data, model also contains the class that manages the model data. To manage the data i.e to retrieve and save employee data we are going to use the following IEmployeeRepository service. At the moment, we only have one method GeEmployee() that retrieves employee by ID. As we progress through the course we will add methods to **Create**, **Update**and **Delete**as well.

public interface IEmployeeRepository  
{  
    Employee GetEmployee(int Id);  
}

The following MockEmployeeRepository class provides the implementation for IEmployeeRepository interface. At the moment, we are hard coding the Employee data in the MockEmployeeRepository class. In our upcoming videos, we will provide another implementation for IEmployeeRepository interface, and that implementation will retrieve data from a SQL Server database.

public class MockEmployeeRepository : IEmployeeRepository  
{  
    private List<Employee> \_employeeList;  
  
    public MockEmployeeRepository()  
    {  
        \_employeeList = new List<Employee>()  
        {  
            new Employee() { Id = 1, Name = "Mary", Department = "HR", Email = "mary@gmail.com" },  
            new Employee() { Id = 2, Name = "John", Department = "IT", Email = "john@gmail.com" },  
            new Employee() { Id = 3, Name = "Sam", Department = "IT", Email = "sam@gmail.com" },  
        };  
    }  
  
    public Employee GetEmployee(int Id)  
    {  
        return this.\_employeeList.FirstOrDefault(e => e.Id == Id);  
    }  
}

Throughout our application we will be programming against the interface IEmployeeRepository and not the concrete implementation MockEmployeeRepository. This interface abstraction allows us to use dependency injection which in turn makes our application flexible and easily unit testable.