Server side Web Development

Lecture

Repetition

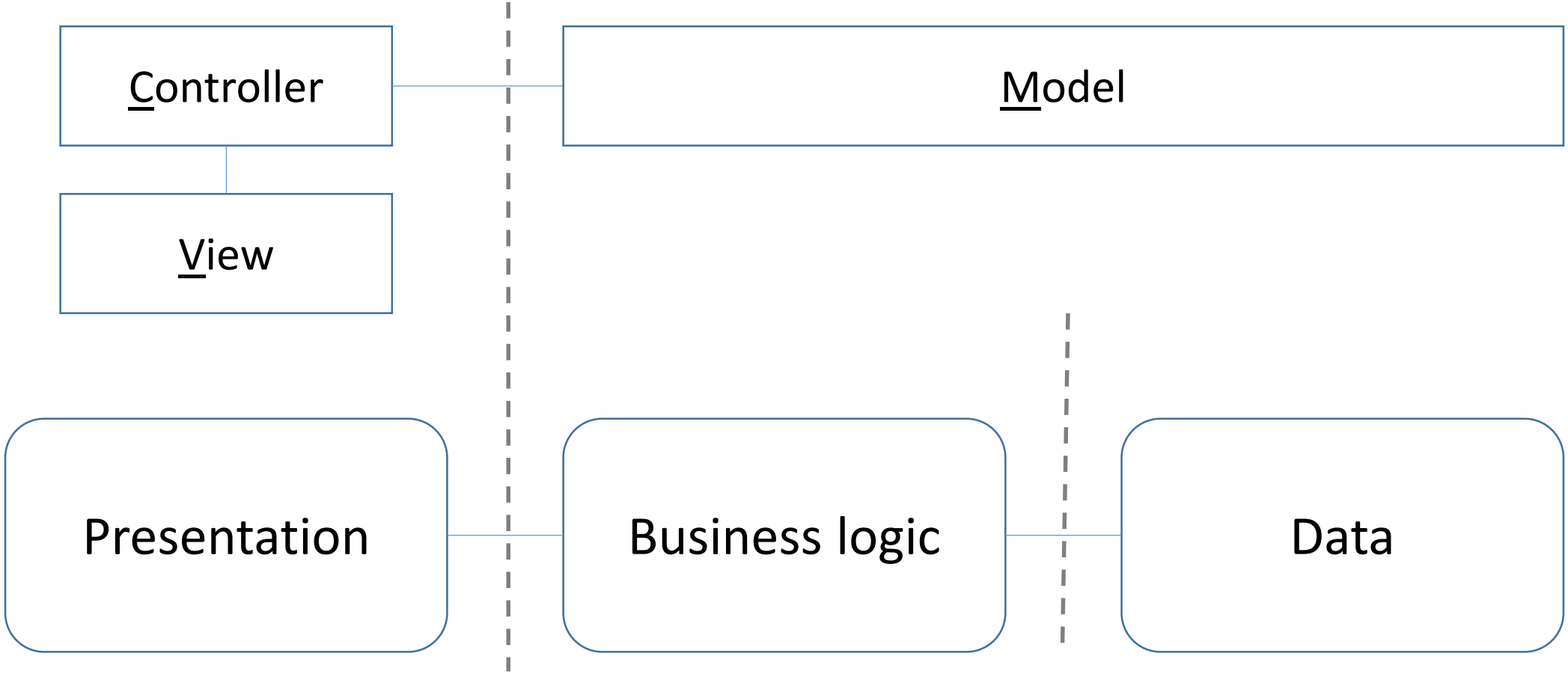
Questions

- When building an ASP.NET MVC web application it is often done by using a three tier architecture. Describe how this usually is done. What are the major benefits?
- What does the acronym MVC stand for in the context of ASP.NET?
  - Model View Controller
- What does the Single Responsibility Principle (also named Separation of Concerns) stand for when it comes to MVC in ASP.NET?

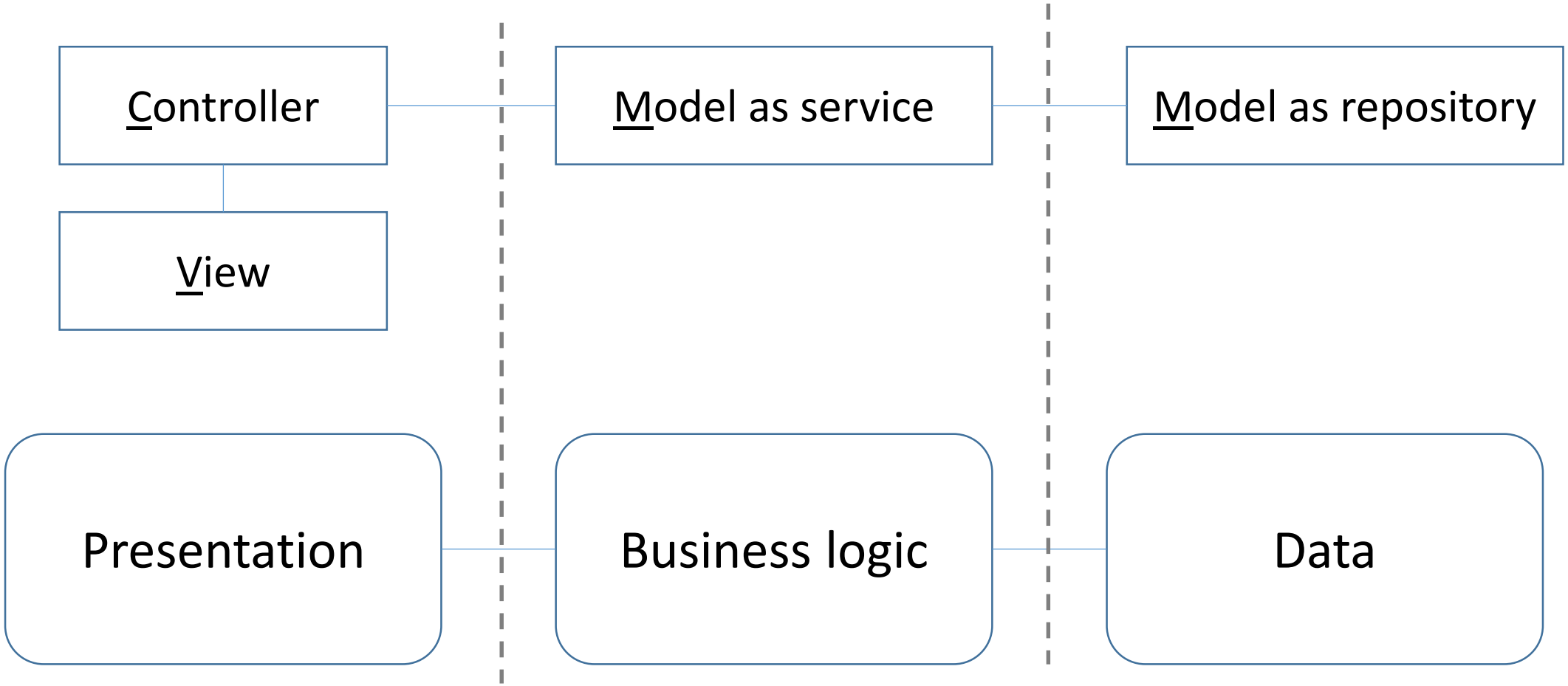
Application layers

- Presentation: the user interface
- Business logic: rules for managing the application context
- Data: persistent storage of data, often in database

Applying the MVC architecture – 2 layers



Applying the MVC architecture – 3 layers



Why divide in three layers?

- Single Responsibility Principle (SRP)
  - A design pattern
  - Each layer should do one and only one thing
  - Independent layers make it easier to:
    - Extend or further develop an application
    - Exchange data providers
    - Connect to multiple data providers (provider model design pattern)
- Requires more initial thinking / work
  - Pays off in large projects
  - Might seem unnecessary in small projects
  - Projects might start small and end up large

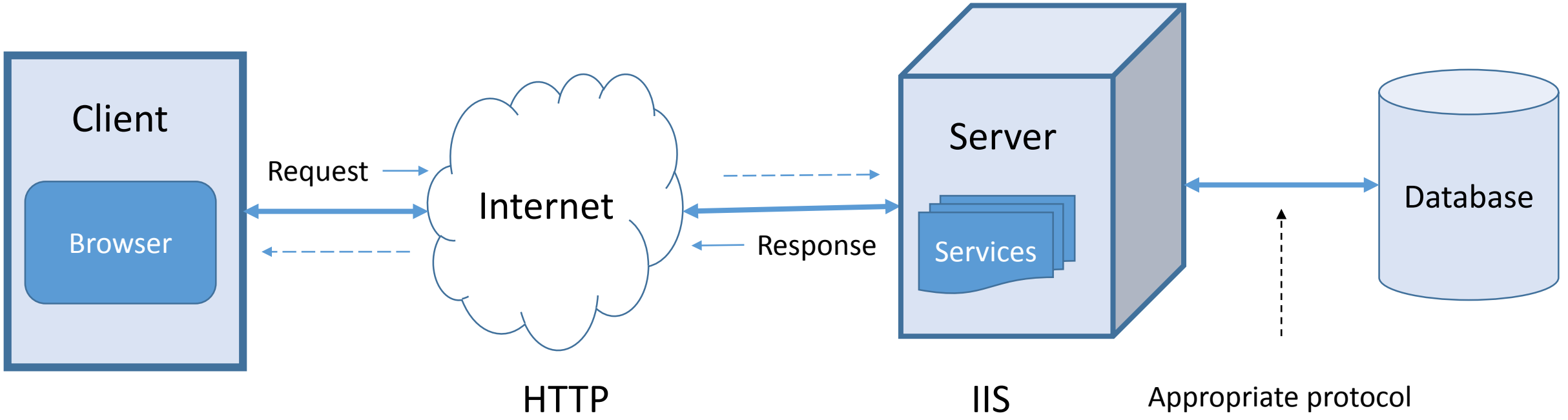
Creating the domain model

- One important concern for the MVC framework is to separate Presentation, Business logic and Data storage
- This is done by introducing several layers (Service and Repository)
  - Often placing these layers in separate projects
  - We will work more with this in coming lectures
- In the initial phase of development it can be useful to do a mockup
  - i.e. to fake data by
  - Adding classes (perhaps not fully outlined) in the Models folder
  - Instantiate objects from these classes and hard-code some data
  - This way you may still work with coding the presentation layer without dealing with the complexity of the database connection

Questions

- Describe the Client/Server web architecture. Give some examples of languages and techniques used in the different parts.
- Describe the purpose of a Controller in an MVC web application.
- When creating a page that will allow users to add products to the product catalog in asp.net MVC, which actions do you take. You can assume that you are using the default routing and folder structure.
  - Add a class to the Controllers folder and name it ProductController, add a method called Add to this class
  - Add a new folder to the Views folder and name it Product. Create a new view in the folder and name that view Add

Web architecture - Client / Server



HTML / JS / CSS

C# / ASP.NET

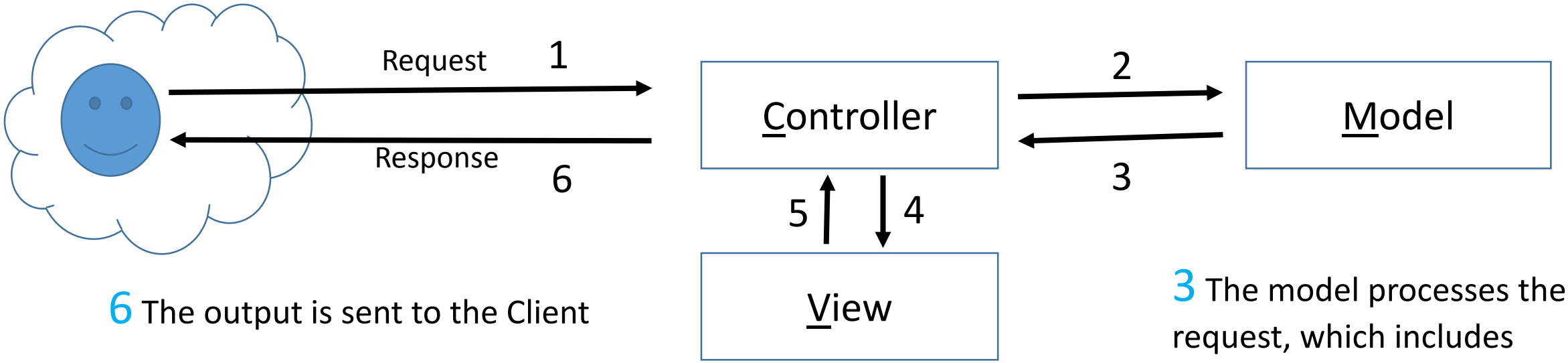
SQL

This is our main focus

Request/Response using MVC architecture

1 Client sends request, ***routed*** to an ***action method*** in a controller class

2 The controller calls methods in the model classes, implementing the business logic



6 The output is sent to the Client

3 The model processes the request, which includes fetching data. Then sends back the result to the controller.

5 The view prepares the output by mixing static and dynamic HTML

4 The controller sends the result to the appropriate view

The controller

- Consists of one or several controller classes
  - Inherits from System.Web.Mvc.Controller
- The controller receives the request
  - Invokes the model for processing the request
  - Selects the appropriate view for preparing the response
  - Passes the response to the client
- The controller has properties for storing information about the current request managed by the controller
  - E.g. HttpContext, Request, Response, RouteData, Session, TempData
- A controller class contains one or several action-methods

ActionResult

- The return value from the action method is an object of the class ActionResult
  - This is not directly the response object sent to the client
  - It describes what the response will be
  - The MVC Framework will process the ActionResult object and produce the response
- There are several sub-classes which describes more specifically the intended response
  - The controller class has helper methods that render these more specific ActionResult objects

Class	Controller method	Description
ViewResult	View	HTML from specified or default view (.cshtml)
PartialViewResult	PartialView	HTML defined in a partial view
RedirectResult	Redirect	Redirect to a URL
RedirectToRouteResult	RedirectToRoute	Redirects to a URL, generated by the routing system
FileResult	File	Returns file-data directly into the response to the browser
ContentResult	Content	Returns a text-string into the response to the browser
JsonResult	Json	Serializes a .NET object in Json format into the response
EmptyResult	null	Does not return anything

The View

- Is where HTML-code is written
- The View-file is given the extension .cshtml (or .vbhtml)
- Most views belong to a certain controller
  - Placed in the Views\<controller name> - folder
- Some views may be shared by several controllers
  - Placed in Views\Shared – folder
- Can be full views or partial views
  - A full view has all parts of an HTML page
  - A partial view has parts of HTML and is included in a full view or another partial view

Questions

- What is the main purpose of using Razor code?
- Explain what a viewbag is and how it can be used?
- What does it mean that a view is strongly typed? Describe the mechanism for passing data to and from a strongly typed view.

Razor code

- Is Written in the View
- Is only meant to unfold the dynamic data content into the view
- Is NOT meant to perform business logic or manipulate the domain model
- All razor code begins with the @ character

Razor code - ViewBag

- A viewbag is a global object
- Properties may be added on the fly in code
- Transfer object value from controller method to a corresponding view
- In the view access the viewbag through razor code
  - @ViewBag.propertyName

Razor - @model

- Declares the type of ***model object*** passed to the View
  - The view needs to be strongly typed
- Allows the View to use the methods, fields and properties of the model object
- E.g. A domain model *Product* with a property *Name* can be accessed in the razor code in a View by

  @Model.Name

- Notice when accessing it a capital M in Model is used

Strongly typed views

- The views need to now how data is represented in different objects
- A strongly typed view uses objects defined by specific class in the models folder
- An action-method associated to this view may pass objects of the class to the view
- The view uses razor syntax to unfold the data from the passed object
- The model object may be also be used to pass data from the view to the controllers action method

Form handling

MVC incudes some helpers for our formhandling:

- BeginForm() - creates the HTML form
- TextBox() - creates an input type text
- Password() - creates an input type password
- RadioButton() - creates a radiobutton
- … for any HTML form input element

Form Handling

@Html.BeginForm(“ActionName”, “ControllerName”, [Method], [RouteValue])

Use @Html.EndForm() to end the form element

or put the BeginForm in a using statement:

```
@using(Html.BeginForm(“Create”, “Author”)){
        //The form
}
```

Form handling

```
public ViewResult AddEmployee()
{
    Employee empObj = new Employee();
    return View(empObj);
}

[HttpPost]
public ViewResult AddEmployee(Employee empObj)
{
    //Some code for storing the data
    return View("ListEmployees",empObj);
}
```

By default the action method is designed for handling a GET request. It is possible to add the attribute [HttpGet], but not necessary

Add the attribute [HttpPost] to capture the POSTED information from the form

Redirection of action method

- POST-requests are used to change the state of an application
  - Issued from a view with a form input, there is a risk that the user will reload the view, thereby resubmitting the form
  - To avoid this problem use the Post-Redirect-Get pattern
  - In the action-method, receiving the posted data, redirect to a GET action-method
- Use *RedirectToAction("action")* in the action-method to redirect internally to another action-method, after the post
- In order to preserve the posted data store it in the TempData variable
  - Similar to how Session variable works, but preserves data only between two consecutive requests

Razor code - Layout

- A layout is a template that can be used in several views
- Creates a common look and feel
- Is created in a seperate View
- A View can set it in its razor code block
  Layout = <The specific layout view file>
- The statement in the razor code block
  Layout = null;
  Means that the view is not using any layout (self contained)
- The layout file name has to begin with an _ (underscore character)

Adding a ViewStart - file

- The ViewStart – file is a layout only containing the line specifying which layout to use
  - I.e.

```
@{
        Layout = "~/Views/Layoutname.cshtml"
}
```

- If a view does not contain a line which specify the layout, the view-engine will look in the ViewStart file for this line
- The purpose is to avoid changing the layout specification on each view if a layout page with a different name is going to be used

Lambda expressions

- A powerful way of writing anonymous functions in a condensed way
  - E.g.
    theSum = (x,y,z) => (x+y+z);
  - Is a function taking three values, stored in the parameters x,y and z, and returning the sum
  - The lambda operator => is interpreted as “goes to”
- Can be used in C# to shortening code writing e.g. when working with Lists

Partial views

- A partial view is a way of reusing HTML and Razor syntax, recurring in several views, by placing this code in a separate partial view
- Will only contain fragments of HTML code and razor syntax
- Does not use layouts
- Create a partial view by adding a view the ordinary way, but check the “create as partial” view option
- Should be placed in the Views/Shared folder
- A partial view that is not strongly typed will by default use the calling views Model

Child Action methods

- Child actions are action-methods called from a within a view
- Controller logic that would be repeated in several views can be placed in a child action method
- Any action method can be used as a child action method
- Usually a child action method would render a PartialView

Questions

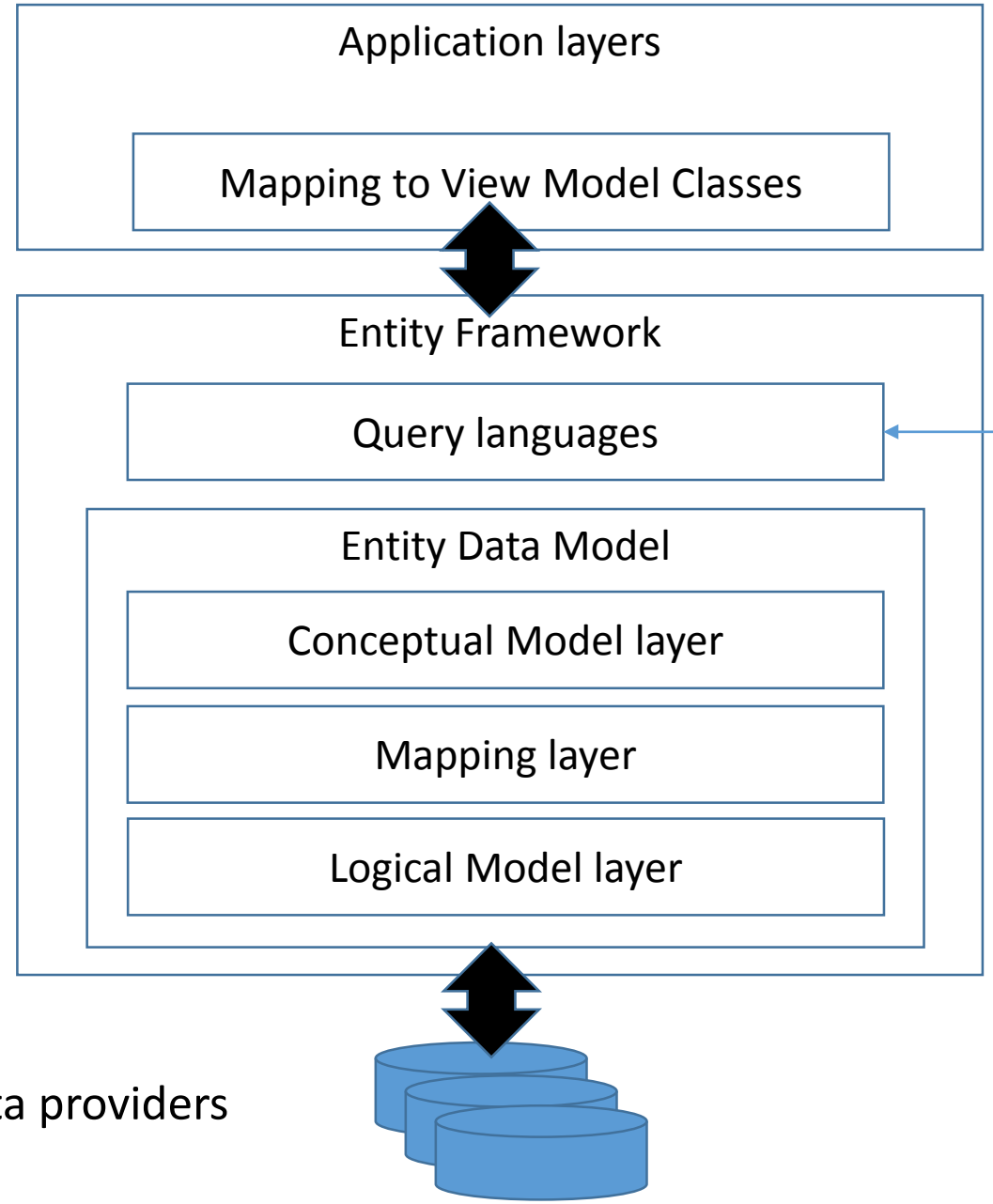
- What is the purpose of using an ORM-tool when building an ASP.NET MVC web application? Give an example of such a tool.

Connecting to the Repository/Data provider

- Pure ADO.NET
  - Traditional old style way, embedding SQL-queries in code and writing the mapping to entity classes
- Object Relational Mapping (ORM) technology
  - Entity Framework (MS) – The dominating technique used today
  - LINQ to SQL (MS)
  - nHibernate (Open Source)

Ways of using Entity Framework

- Different ways to work with Entity Framework (EF):
  - Dabase first
    - Through visual studio the EF designer will use an already existing database to generate Entity classes and code for managing data to/from the database
    - Manual changes of the database is supported
  - Model first
    - The EDM designer tool in Visual Studio is used to model the database
    - EF uses the model to create the database
  - Code first
    - The domain model classes and datamappings are defined first
    - EF uses Model Classes and database mappings to create the database



Linq to SQL
Entity SQL
DBContext (Wrapper of Objectcontext)

Mapping entity and view models

- The entity classes are different from view model classes
- It is necessary to map data loaded in objects created from entity classes to objects created from view model classes
- Writing code for this manually is elaborate and tedious
- Tools for automatic mapping is available e.g. Automapper

Querying the conceptual model

- EF provides an API called DbContext
- Installing EF creates a class derived from DbContext
- To query the database an instance of this class is required
- Queries can be formulated using LINQ
  - Language Integrated Query
- Usually LINQ – queries are embedded in methods, which then are called by the service layer

Local vs the real database

- Queries using DbContext can be run locally
  - This means it uses already loaded data, if possible
  - Pros:
    - It goes faster to work with loaded data
    - It is possible to make several changes before applying them to the database
    - It is possible to run queries even if the application is detached from the database
  - Cons:
    - The real data in database may have changed from what is locally available
    - Might cause inconsistencies in database

Loading schemes (referenced data)

- Basically three types
  - Lazy loading
  - Eager loading
  - Explicit loading