**3- State 3 cases that ViewModel is a must and one of this cases Security case.**

**1. Combining Data from Multiple Models**

When you need to display or process data from multiple models in a single view, a ViewModel is necessary to combine the data into a single object. For example, if you want to display a department along with a list of students and courses, you can create a ViewModel that includes properties for the department, students, and courses.

**2. Customizing Data for the View**

Sometimes, the data you need to display in a view is not a direct representation of your model. You might need to format the data, calculate additional properties, or include only a subset of the model's properties. A ViewModel allows you to shape the data specifically for the view. For example, you might want to display a department's name and a list of student names over 25 years old, along with the department state based on the student count.

**3. Security: Preventing Over-Posting Attacks**

A ViewModel can help prevent over-posting attacks by limiting the properties that can be bound from user input. When you use a ViewModel, you can ensure that only the properties you intend to update are included in the model binding process. This prevents malicious users from attempting to update properties that should not be modified. For example, if you have a Department model with sensitive properties like Id or MgrName, you can create a ViewModel that excludes these properties to ensure they cannot be modified through form submissions.

EX  
Consider a Department model with sensitive properties:

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

public string MgrName { get; set; }

public List<Student> Students { get; set; }

}

To prevent over-posting attacks, create a ViewModel that excludes the Id and MgrName properties:  
public class DepartmentViewModel

{

public string Name { get; set; }

public List<string> StudentNamesOver25 { get; set; }

public string DepartmentState { get; set; }

}  
  
In the controller, use the ViewModel for binding:

public IActionResult Add(DepartmentViewModel viewModel)

{

if (ModelState.IsValid)

{

var department = new Department

{

Name = viewModel.Name,

Students = // map students from viewModel.StudentNamesOver25

};

services.Add(department);

return RedirectToAction("ShowAll");

}

return View(viewModel);

}

By using a ViewModel, you ensure that only the intended properties are updated, enhancing the security of your application.

# [**4- LinkedIn article about Filter,Model Binding**](https://www.linkedin.com/posts/ahmed-b-ramzy_filters-model-binding-in-aspnet-core-activity-7300222748851605505-kupf?utm_source=share&utm_medium=member_desktop&rcm=ACoAADuIxXkBdVPXITPJTt0yTVu1y8HgMAABtYE)