Western Norway
University of
Applied Sciences

DAT152 – Advanced Web Applications

Backend Web Development

Agenda @Today

- Backend web development
- Strategies for MVC (Revision)

Architecture or design?

- Architectural pattern: High-level structure of software system
  - MVC
  - Layer
  - Pipeline
  - Microservices
  - Client-Server
  - etc
- Design pattern: Low-level design of individual components/modules/classes
  - Singleton pattern
  - Decorator
  - Factory pattern
  - Command pattern
  - Builder pattern
  - etc

Example - Decorator

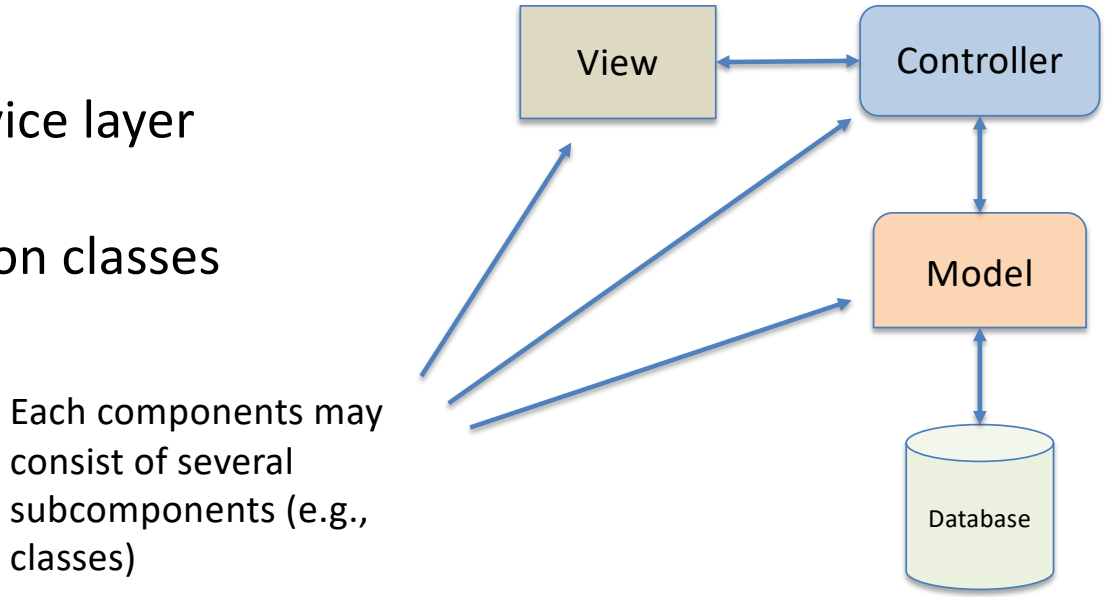
- Using authenticator/authorization filters in Frameworks for intercepting HttpRequest

MVC Architectural Pattern

MVC = Model View Controller

MVC pattern consists of three parts:

- Model: Domain object model/service layer
- View: Template code/markup
- Controller: Presentation logic/action classes



Why use MVC?

https://www.tomdalling.com/blog/software-design/model-view-controller-explained/

MVC Architectural Pattern

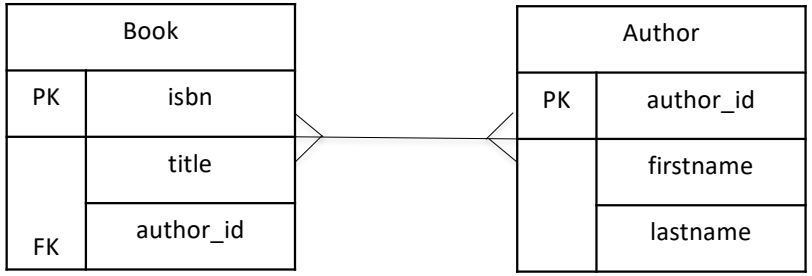
- Responsibility-driven design
  - How to assign responsibility to objects
  - What role objects should play in a collaboration
- Single Responsibility Problem:
  - every object in your system should have a single responsibility, and all the object's services should be focused on carrying out that single responsibility.

Model

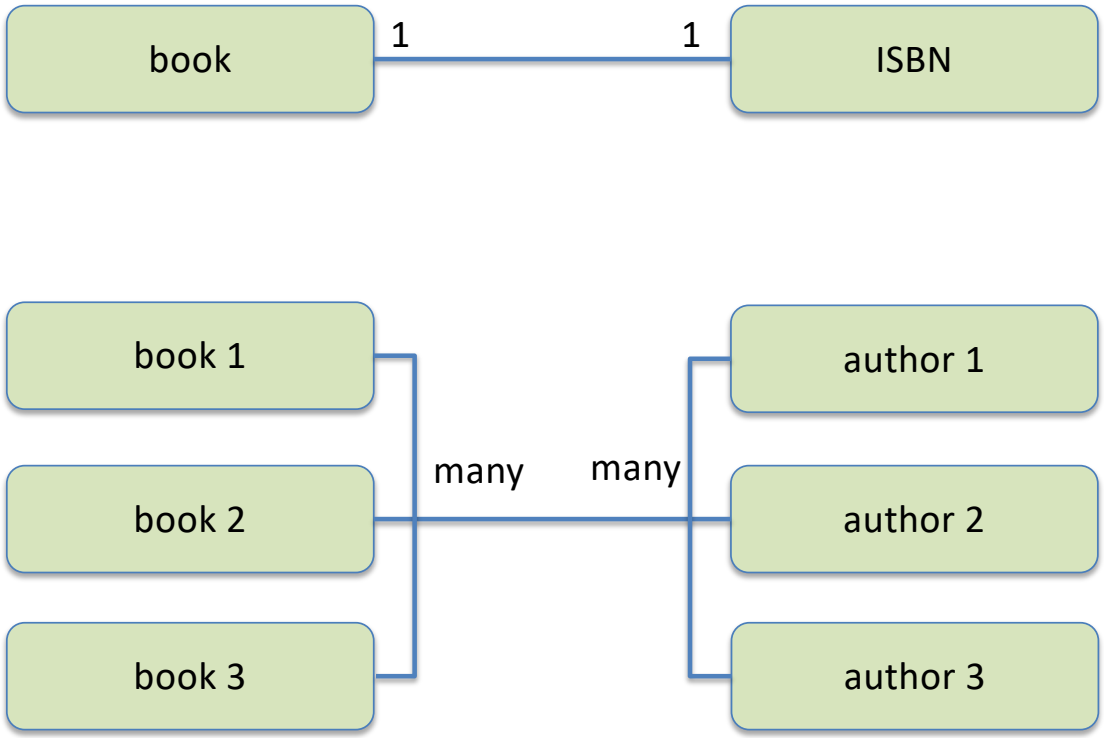
- Models the actual problem being solved
- Domain objects
- Independent of both the controller and the view
- Provides flexibility and robustness

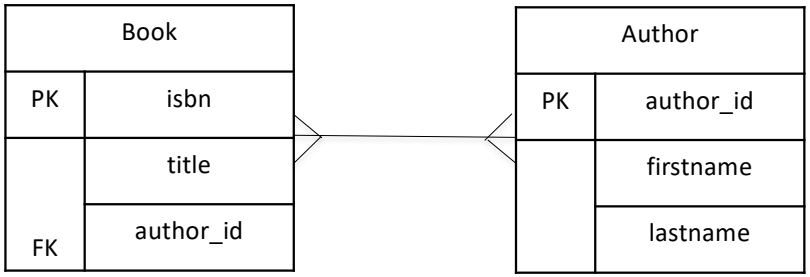
Problem: Create a library service

H2
MySQL
Postgres
Oracle
MSSQL



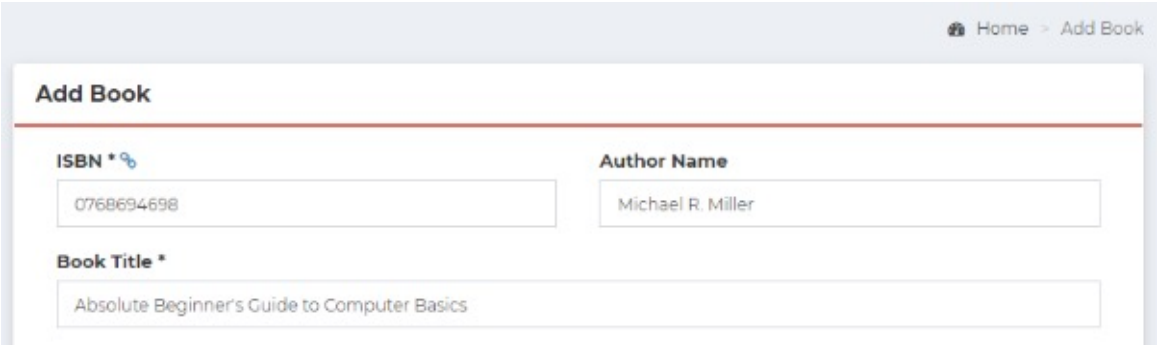
Model


book
1
1
ISBN
book 1
book 2
book 3
many
many
author 1
author 2
author 3


Book
PK
isbn
title
author_id
FK
Author
PK
author_id
firstname
lastname

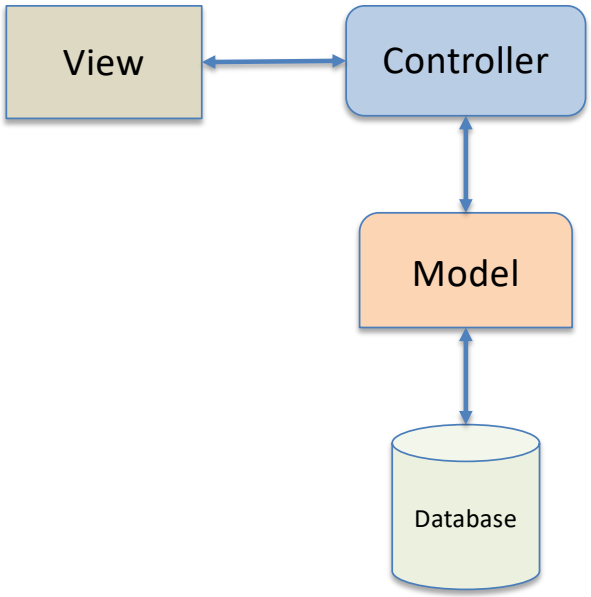
View

- Responsible for user interface (UI)
- User needs to see or view what the program is doing
- So, asks model for data and presents it in a user-friendly format



Controller

- Translates UI actions into operations on domain objects (model)
- Decides what the model does
- Design of controller depends on the model
- Model should not depend on the controller



Advantages

- Eases maintenance
  - Easy to add, change, or modify components (e.g., add new view)
  - Loosely coupled architecture
- Easier to understand components individually
- Easier to test components independently (unit testing)
- Separation of responsibilities between programmers and designers

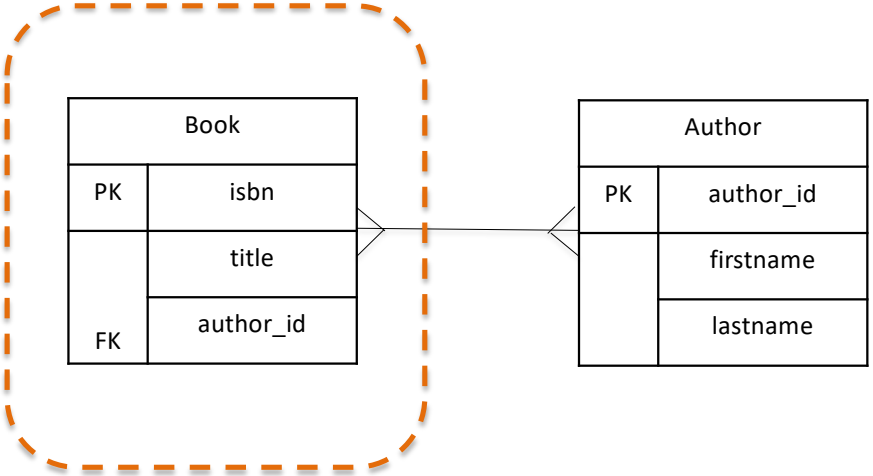
Java Backend Web Technology

- Java EE
  - Java/Jakarta Server Page (JSP)
  - Java/Jakarta Server Faces (JSF)
  - Servlets

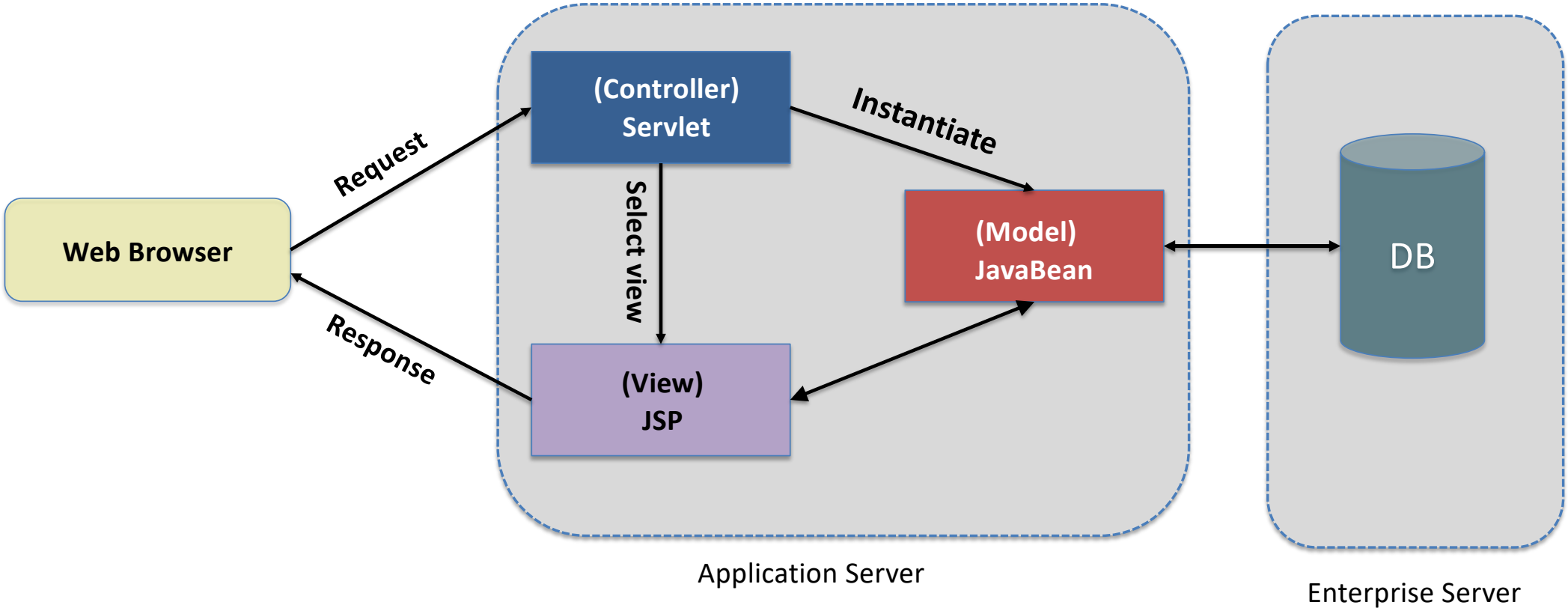
- Spring Framework

MVC (without framework)

- We will build a book repository (like a library service)
- Java EE
  - Model
    - JavaBeans +
    - We need a database for persistence
      - e.g., Derby/H2/MySQL/Postgres)
  - Controller
    - Servlets
    - Configure in web.xml
  - View
    - JSP
    - HTML



MVC



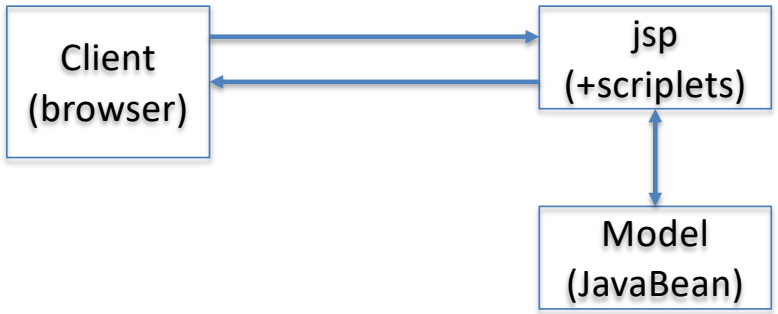
https://www.oracle.com/technical-resources/articles/javase/servlets-jsp.html

Some design options

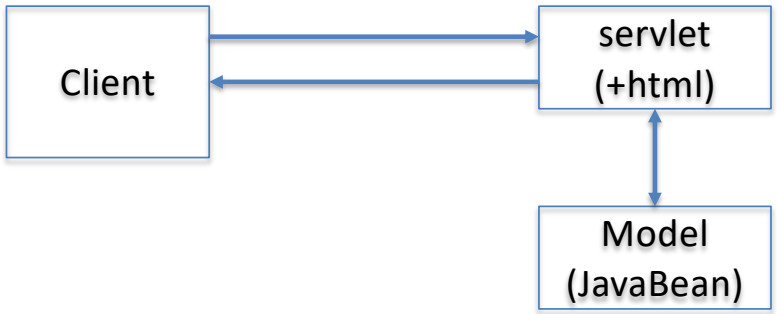
- Several controllers
- Front Controller pattern
- Command pattern
- Flow Manager

Anti-pattern MVC

- MVC pattern can be implemented in ways that violate software quality attributes (Maintainability, reusability, extensibility, etc)



View + controller



Controller + view

Anti-pattern MVC

- Mixing application logic and markup is bad practice
  - Violates single responsibility principle
  - Harder to change and maintain
  - Error prone
  - Harder to re-use

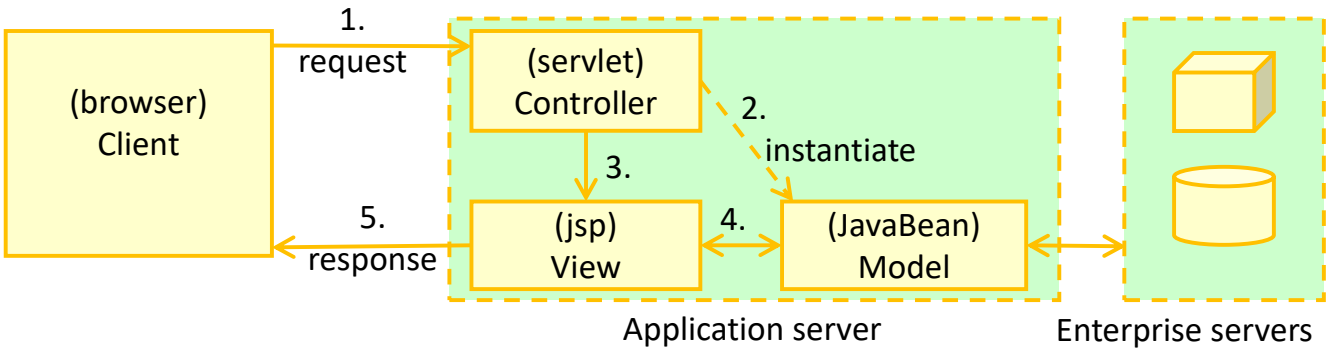
We will not talk about this anymore

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Insert title here</title>
</head>

<body>
<%= String name=request.getParameter("username");
out.print("Hello "+name);
%>
</body>

</html>
```

View templates such as Thymeleaf is introduced to avoid mixing business logic with the view

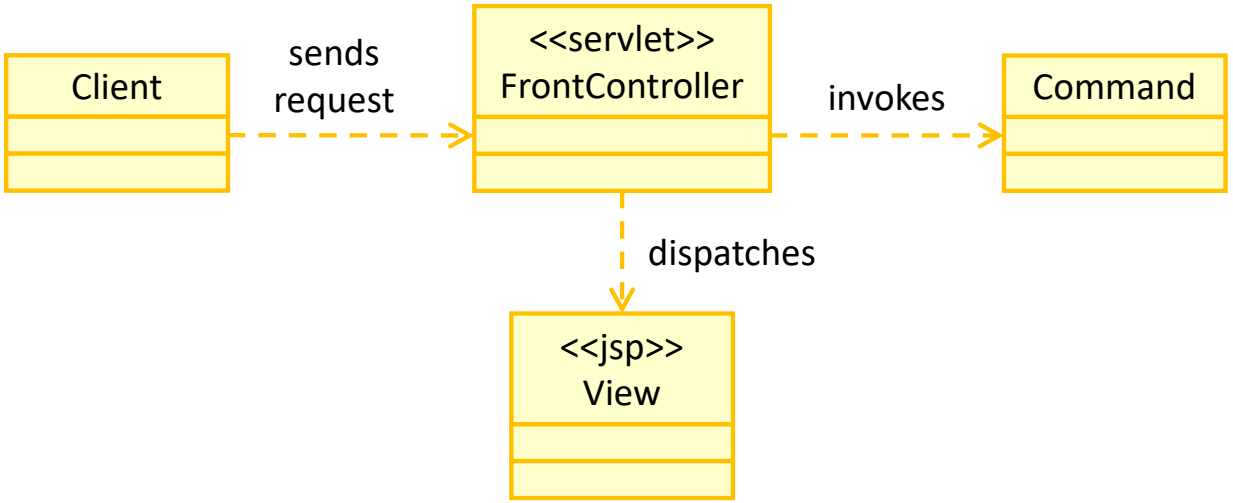
Several controller MVC

- Make MVC for each use case
- Use multiple controller servlets
- Request go directly to the controller responsible



FrontController MVC

- FrontController is a design pattern dealing with centralization of processing of requests and selection of views in a single component
  - The application gets a single access point where all requests go through.
  - The request one wants is provided either as part of the URL, or as parameters in the request.



FrontController MVC

- In a Java web app, the Front Controller is usually a Servlet (can also be a filter), and often the only Servlet
- In the FrontController, we can place all the key tasks (e.g., selection of actions and views)

FrontController MVC

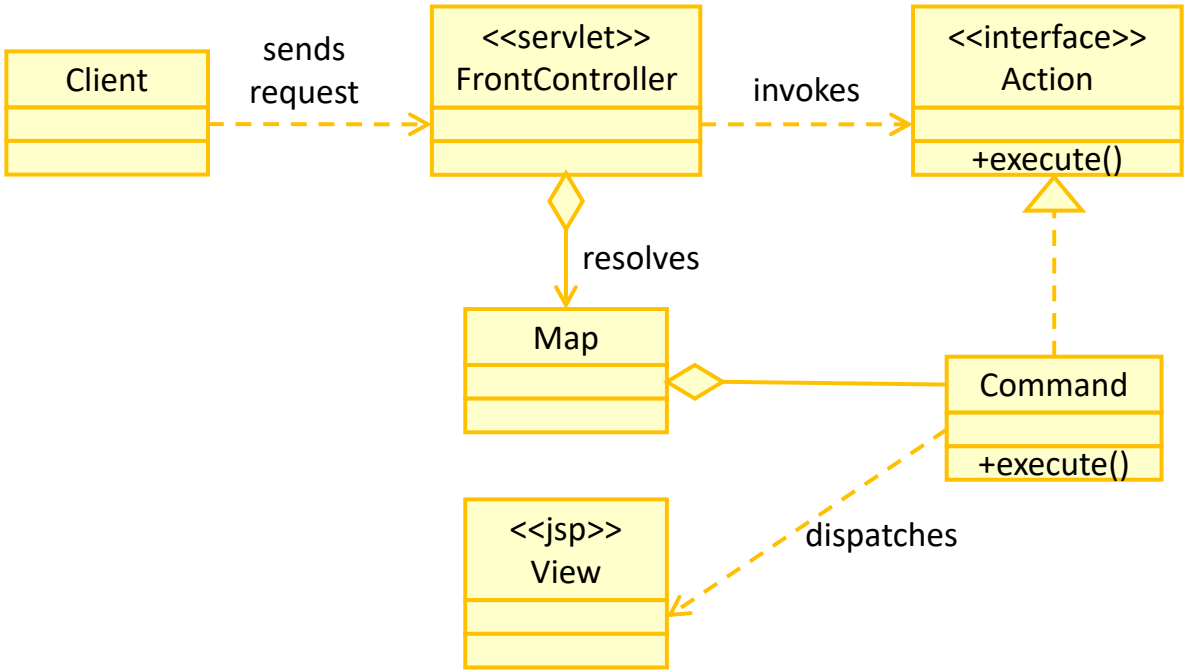
- Naive FrontController
  - Using if-else to decide what action?
  - Problem: controller grows as system evolves

```
if (cmd.equals (”viewbook")) {
        ... Doing all the work here
} else if (cmd.equals(”addbook")) {
    ... Doing all the work here
} else if ... etc.
```

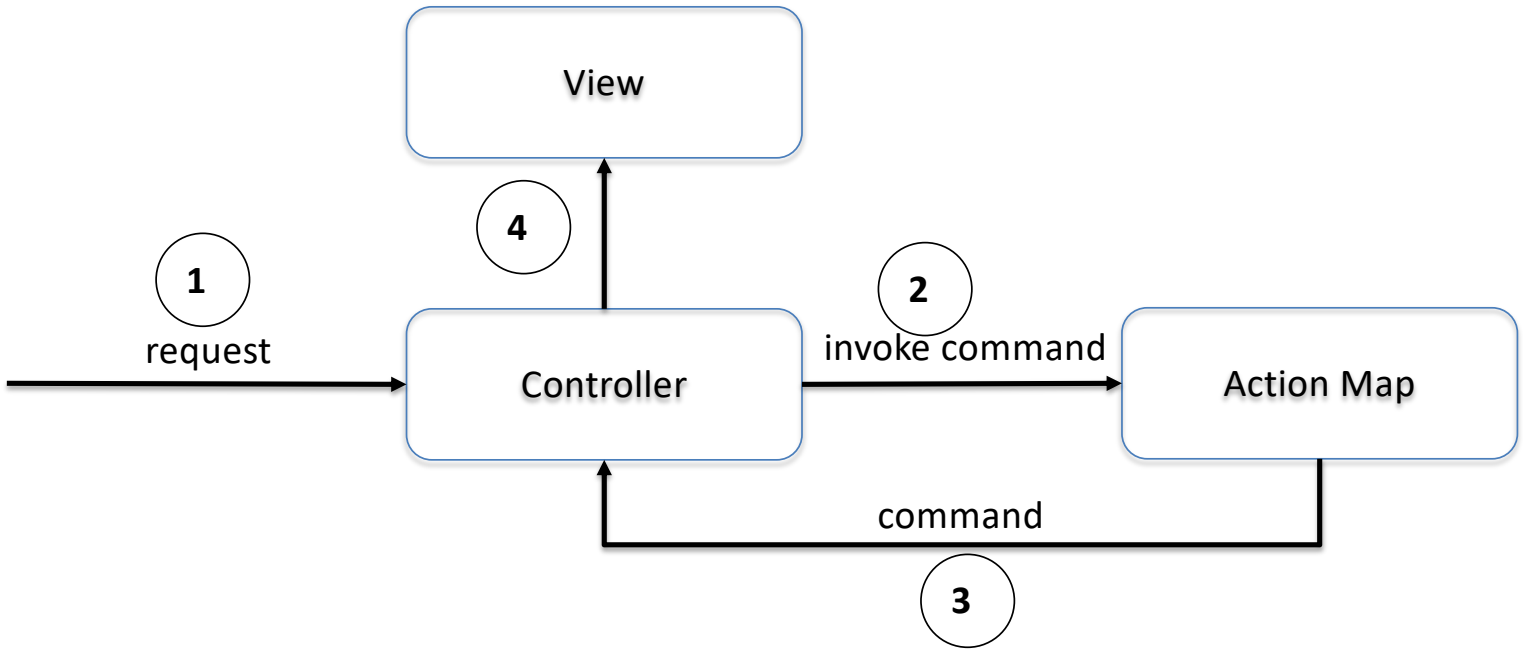
Demo

FrontController MVC

- Command pattern
  - Better solution
  - Encapsulate a command (e.g., addBook) with associated data and business logic as an object
  - Use “polymorphism” instead of “if-else” to invoke the right command



FrontController: Command Pattern

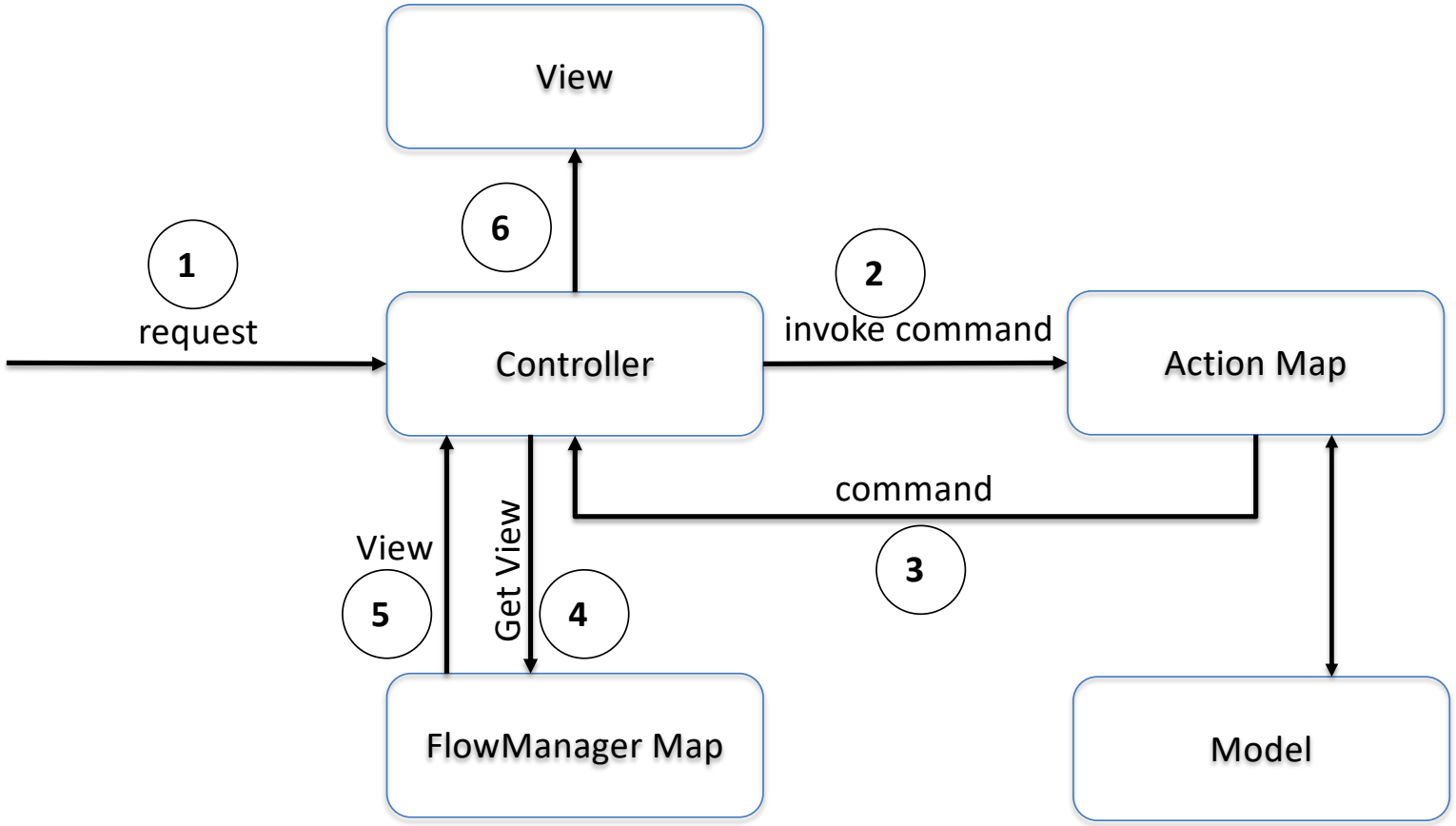


Demo

FrontController MVC

- Command dispatching is centralized
- However,
  - Control of page flow is still decentralized control (every action determines what the next page is).
  - It could be desirable to gather information about the page flow in a centralized place.
- We can use FlowManager to keep track of page flow
  - Command pattern + FlowManager

FrontController With FlowManager



Demo

Summary

- Dynamic page flow
- FrontController: Centralized control and common logic
- Command Pattern: Business logic in regular classes
- FlowManager: Centralized description of page flow
  - Automatic action mapping -> Does not need programmatic logic for mapping between requests and actions. Along with other things is FrontController now completely general, and eliminating the need for being changed by the developers.

Need for Web Development Framework?

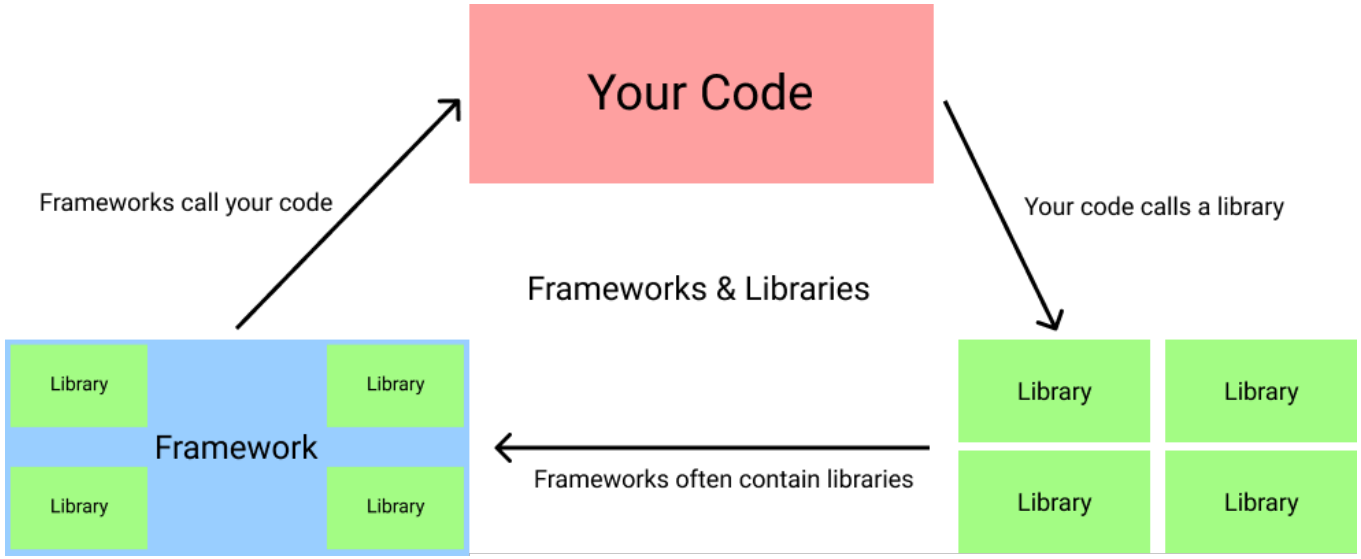
- The reason why there are frameworks for web development is to simplify development
- Through actions that we have just seen, the whole Servlet API can be camouflaged, and what we want done is more straightforward,
- Framework will also often offer:
  - Easy and flexible ways to configure things (via XML or Annotations)
  - Tag library, for example validating input (Spring)
  - Class Library
  - ++

What is a framework?

- a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

https://en.wikipedia.org/wiki/Software_framework

Framework vs. Library



https://designenterprisestudio.com/2022/05/26/libraries-frameworks/

Why use frameworks?

- Web development is complex
- Good to have a common architecture that everyone follows
- Get support on part of the common functionality (i18n, validation, authentication and authorization, request processing, assembly of modules, ...)
- Less own code to maintain
- Faster and more robust development
- Easier to test
- Community support

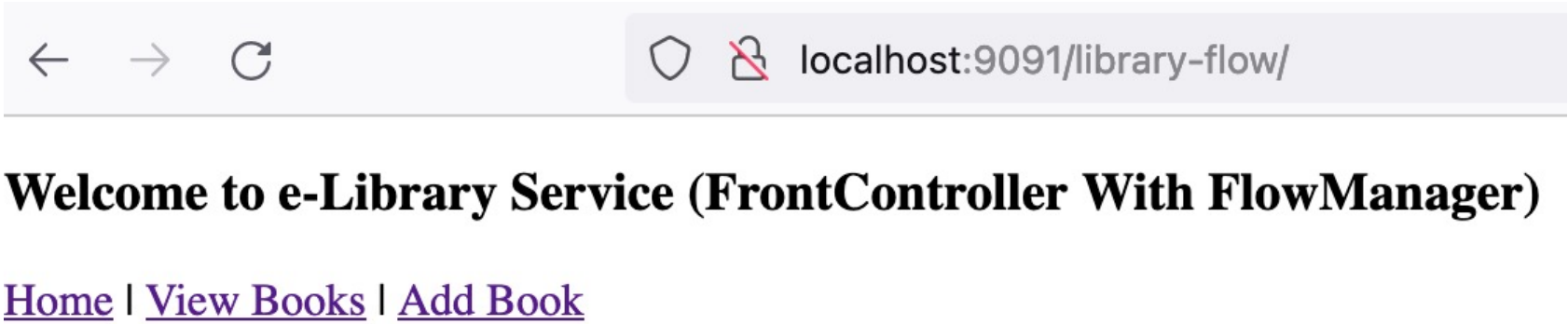
Cons?

- Lots to choose from
- Dependence / "lock-in"
- What if the framework "dies"?

Next

- Web Development Frameworks
  - Spring Web MVC

Lab Exercises



New Features

- Add Author
- Delete Book

Extra challenge (++)

Security Features?

- Authentication
- Authorization