

Web technologies

Architectural pattern
Model-View-Controller

David Jelenc

Index

- 1 Architectural Patterns
 - Architectural and Design Patterns
 - Model-View-Controller
 - MVC Frameworks

Index

- 1 Architectural Patterns
 - Architectural and Design Patterns
 - Model-View-Controller
 - MVC Frameworks

The purpose of patterns

- Software systems are complex;

The purpose of patterns

- Software systems are complex;
- A pattern represents an implementation plan for our application;

The purpose of patterns

- Software systems are complex;
- A pattern represents an implementation plan for our application;
- A pattern describes how to break the implementation into distinct modules, and assign responsibilities and interactions between those modules;

The purpose of patterns

- Software systems are complex;
- A pattern represents an implementation plan for our application;
- A pattern describes how to break the implementation into distinct modules, and assign responsibilities and interactions between those modules;
- The goal is to **improve the organization of our code.**

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;
 - **View** presents the data to the user;

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;
 - **View** presents the data to the user;
 - **Controller** receives user input and orchestrates the first two;

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;
 - **View** presents the data to the user;
 - **Controller** receives user input and orchestrates the first two;
- Usually in web applications, we add a **Router**:

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;
 - **View** presents the data to the user;
 - **Controller** receives user input and orchestrates the first two;
- Usually in web applications, we add a **Router**:
 - Represents **a single entry point** to the application that **decides which controller** should be executed for each request;

Model-View-Controller (MVC)

- The goal: **a clear separation between business logic and presentation.**
- Invented in 1979, not adjusted for the web
- Three modules:
 - **Model** handles data and *business logic*;
 - **View** presents the data to the user;
 - **Controller** receives user input and orchestrates the first two;
- Usually in web applications, we add a **Router**:
 - Represents a **single entry point** to the application that **decides which controller** should be executed for each request;
 - (The router can be seen as part of the controller.)

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);
- **View:** Classes and scripts that define the output: HTML, XML, JSON and other types;

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);
- **View:** Classes and scripts that define the output: HTML, XML, JSON and other types;
- **Controller:** Classes and scripts that handle HTTP requests, parse parameters, and validate input on the server-side.

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);
- **View:** Classes and scripts that define the output: HTML, XML, JSON and other types;
- **Controller:** Classes and scripts that handle HTTP requests, parse parameters, and validate input on the server-side.
 - The controller invokes a model, gets data from it, and passes it to a view, which draws the response;

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);
- **View:** Classes and scripts that define the output: HTML, XML, JSON and other types;
- **Controller:** Classes and scripts that handle HTTP requests, parse parameters, and validate input on the server-side.
 - The controller invokes a model, gets data from it, and passes it to a view, which draws the response;
 - The controller is the mediator (the *middle man*) between the model and the view.

MVC in a web applications: a typical example

- **Model:** Classes and scripts that handle the business part of the application (usually the database);
- **View:** Classes and scripts that define the output: HTML, XML, JSON and other types;
- **Controller:** Classes and scripts that handle HTTP requests, parse parameters, and validate input on the server-side.
 - The controller invokes a model, gets data from it, and passes it to a view, which draws the response;
 - The controller is the mediator (the *middle man*) between the model and the view.
- **Router:** A module that maps a URLs to controllers.

MVC: The overview of invocations

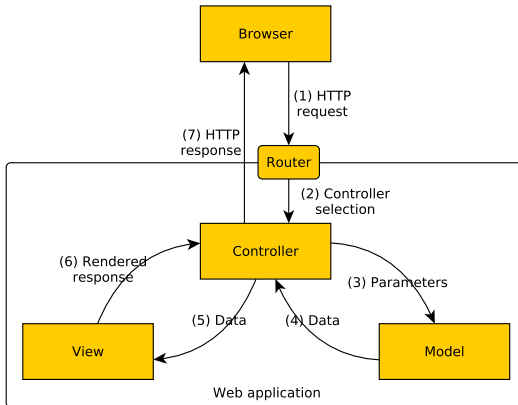


Figure: Invocation sequence

MVC: Advantages and disadvantages

Advantages

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)
- Code artifacts are easier to locate and change (increased ease of modification)

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)
- Code artifacts are easier to locate and change (increased ease of modification)
- Models can have multiple views (better code reuse)

Disadvantages

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)
- Code artifacts are easier to locate and change (increased ease of modification)
- Models can have multiple views (better code reuse)

Disadvantages

- Increased project structure complexity (additional levels of abstraction)

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)
- Code artifacts are easier to locate and change (increased ease of modification)
- Models can have multiple views (better code reuse)

Disadvantages

- Increased project structure complexity (additional levels of abstraction)
- Decomposing features into multiple building blocks means that **a single action** is scattered into multiple locations

MVC: Advantages and disadvantages

Advantages

- Related actions are grouped together (improved code cohesion)
- Modules have well-defined tasks and interfaces (reduced coupling)
- Code artifacts are easier to locate and change (increased ease of modification)
- Models can have multiple views (better code reuse)

Disadvantages

- Increased project structure complexity (additional levels of abstraction)
- Decomposing features into multiple building blocks means that a **single action** is scattered into multiple locations
- Requires a bit of practice (non-negligible learning curve)

MVC Frameworks

- There are several good and open-source MVC frameworks:

MVC Frameworks

- There are several good and open-source MVC frameworks:
 - Laravel, Symfony, Zend, CakePHP, CodeIgniter ...

MVC Frameworks

- There are several good and open-source MVC frameworks:
 - Laravel, Symfony, Zend, CakePHP, CodeIgniter ...
 - The best way to learn about the MVC pattern, is to make your own framework!

MVC Frameworks

- There are several good and open-source MVC frameworks:
 - Laravel, Symfony, Zend, CakePHP, CodeIgniter ...
 - The best way to learn about the MVC pattern, is to make your own framework!
- **When making a real PHP application, one would typically use one of the well-known and tested frameworks!**