Lecture 6.1
Application Frameworks
Model View Controller(MVC) Design Pattern
LARAVEL

Course Instructor

Engr. Madeha Mushtaq

Software Application Framework

- Evolution of programming and where Software frameworks fit in.

Software Application Framework

- Most operating systems have a software framework for developing applications that will run in their environment.
- Microsoft has the windows presentation foundation, framework for building windows based applications.
- Apple has cocoa, Linux has norm or KDE framework.

Software Application Framework

- A software application framework is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications and products.

Software Application Framework

- A software application framework can be considered as consisting of two parts:
- **Application Frameworks – Frozen Spots:**
- The parts that you generally don't change in any instantiation of the framework.
- Define the overall architecture – the basic components and relationships between them. The application infrastructure ("plumbing").

Software Application Framework

- **Application Frameworks – Hot Spots:**
- The parts where programmers are supposed to add their own code.
- The means through which you extend the behavior of the framework – this is what gives the final application its specific functionality.

Software Application Framework

- **Obvious Benefit:**
- Programmers get a lot of functionality they don't have to think about (frozen spots), and can focus their creativity on the unique requirements of their application.
- Don't fight the framework!

Web Application Framework

- An application framework designed to support development of web applications that generally includes:
  - Database Support
  - Templating Framework for generating dynamic web content.
  - HTTP session management and middleware support.
  - Built-in testing framework.

Web Application Framework

- Some popular web application frameworks:
- Laravel (Symfony, PHP)
- Ruby on Rails (Ruby)
- Play (Java snd Scala)
- ASP.NET MVC (Microsoft)
- Django (Python)
- Sinatra (Ruby)
- Symfony (PHP)
- Sails.js (Node.js, Javascript)

MVC Design Pattern

- Early web application architectures did not have much organization on the server side:
  - Primarily fetching static web pages.
  - No separation of data from its presentation.
  - As applications became richer, server-side scripts became more complicated and the applications became very difficult to maintain.

MVC Design Pattern

- The Model-View-Controller (MVC) design pattern provides a means for dealing with this complexity:
  - Decouples data (model) and presentation (view).
  - A controller handles requests and coordinates between the model and the view.
  - More robust applications, easier to maintain.

Parts of MVC

- A Model View Controller pattern is made up of the following three parts:
  - Model
  - View
  - Controller

Model

- The model is responsible for managing the data of the application.
- It responds to the request from the view and it also responds to instructions from the controller to update itself.
- The Model represents the application core (for instance a list of database records).

Model

- Models are based on real-world items such as a person, bank account, or product.
- E.g. If you were building a blog, your models might be post and comment.
- A model is more than just data; it enforces all the business rules that apply to that data.
- For example, if a discount shouldn't be applied to orders of less than $10, the model will enforce the constraint.

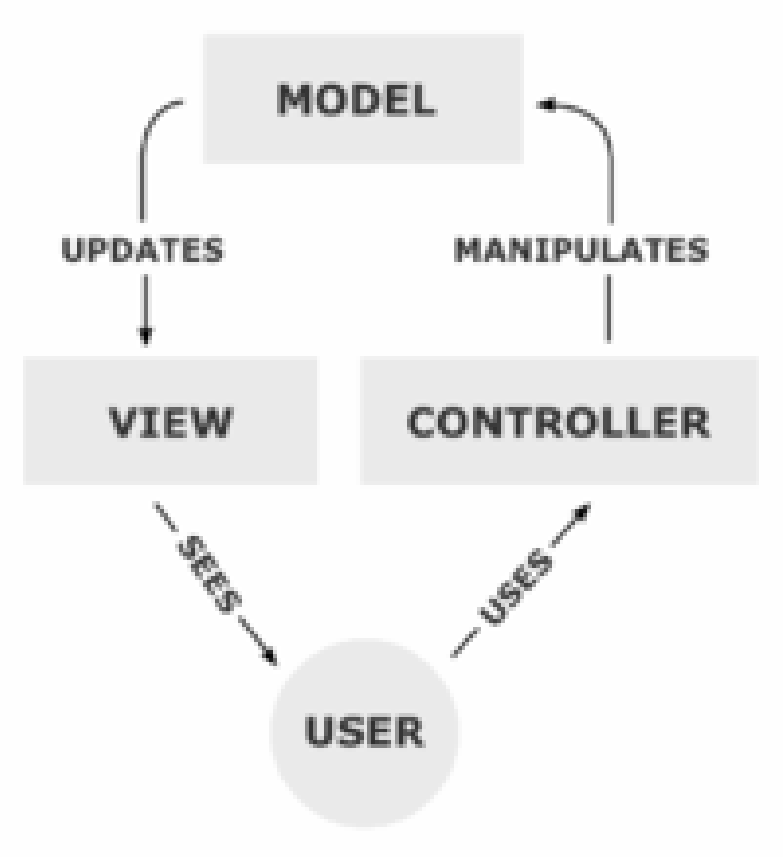
View

- The View displays the data (the database records).
- The visual representation of a model, given some context.
- A view requests information from the model, that it needs to generate an output representation.
- It's usually the resulting markup that the framework renders to the browser, such as the HTML representing the blog post.
- The view layer is responsible for generating a user interface, normally based on data in the model.

Controller

- The Controller is the part of the application that handles user interaction.
- The coordinator that provides the link between the view and the model.
- Typically controllers read data from a view, control user input, and send input data to the model.
- It handles the input, typically user actions and may invoke changes on the model and view.

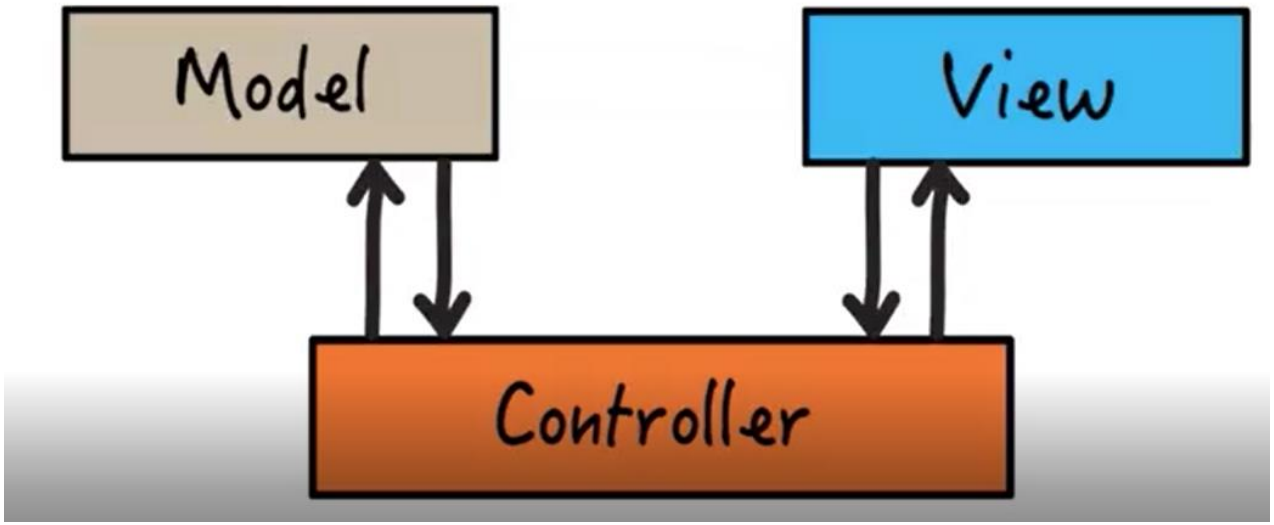
Workflow in MVC



Workflow in MVC

Before MVC:

Using MVC:



Workflow in MVC - Example

- Though MVC comes in different flavours, the control flow generally works as follows:
- The user interacts with the user interface in some way (e.g., user presses a button)
- A controller handles the input event from the user interface, often via a registered handler or callback.
- The controller accesses the model, possibly updating it in a way appropriate to the user's action (e.g., controller updates user's shopping cart).

Workflow in MVC - Example

- A view uses the model to generate an appropriate user interface (e.g., view produces a screen listing the shopping cart contents).
- The view gets its own data from the model.
- The model has no direct knowledge of the view.

Dependence Hierarchy

- There is usually a kind of hierarchy in the MVC pattern.
- The Model knows only about itself.
- That is, the source code of the Model has no references to either the View or Controller.

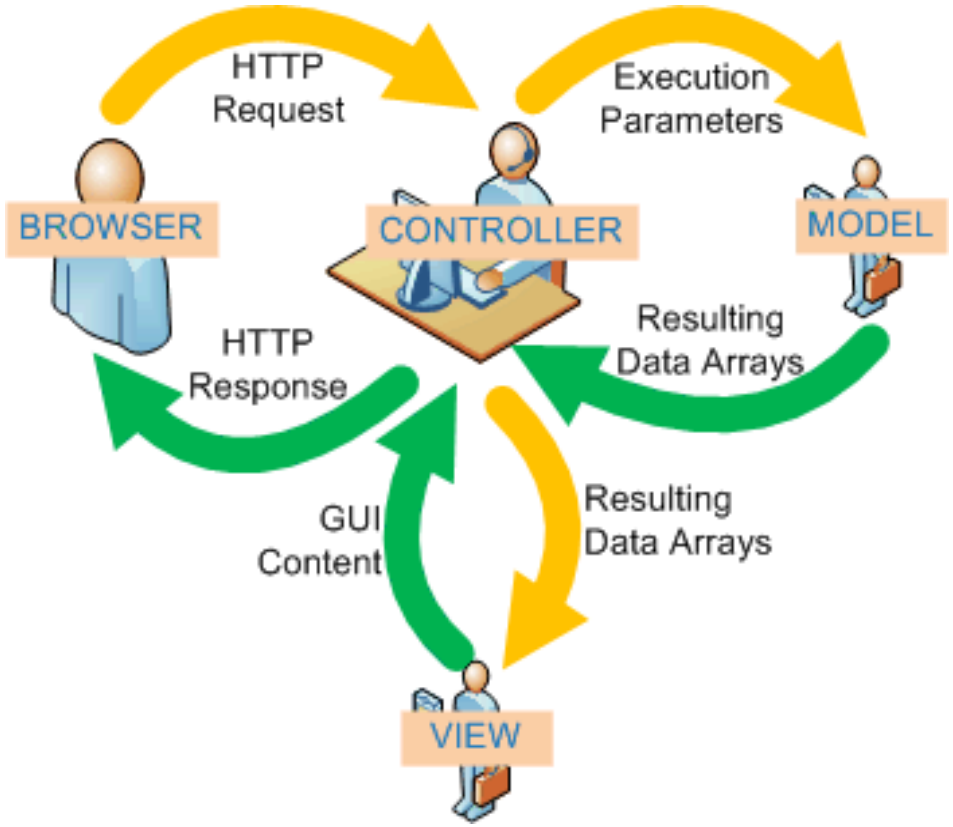
Dependence Hierarchy

- The View however, knows about the Model. It will poll the Model about the state, to know what to display.
- That way, the View can display something that is based on what the Model has done.
- But the View knows nothing about the Controller.
- The Controller knows about both the Model and the View.

Why Dependence Hierarchy?

- The reason to keep it this way is to minimize dependencies.
- No matter how the View class is modified, the Model will still work.
- Even if the system is moved from a desktop operating system to a smart phone, the Model can be moved with no changes.
- But the View probably needs to be updated, as will the Controller.

Working of MVC in Web Application



Advantages

- Clear separation between presentation logic and business logic.
- Each object in mvc have distinct responsibilities.
- parallel development
- easy to maintain and future enhancements

Laravel

- Laravel is a Full-Stack PHP web framework.
- The Laravel follows the model-view-controller (MVC) architectural pattern which enforces a separation between “business logic” from the input and presentation logic associated with a graphical user interface (GUI).
- It is universal re-usable platform to develop web applications.
- It has many features such as session management, database management, Composer and many more.

Laravel Installation

- First Install Composer
- Step 1:

  Open www.getcomposer.org
- Step 2:

  Click on Download button

What is Composer?

- Composer is a software, that manages PHP dependencies that is package management system and package/class auto loader.
- It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.

Laravel Installation

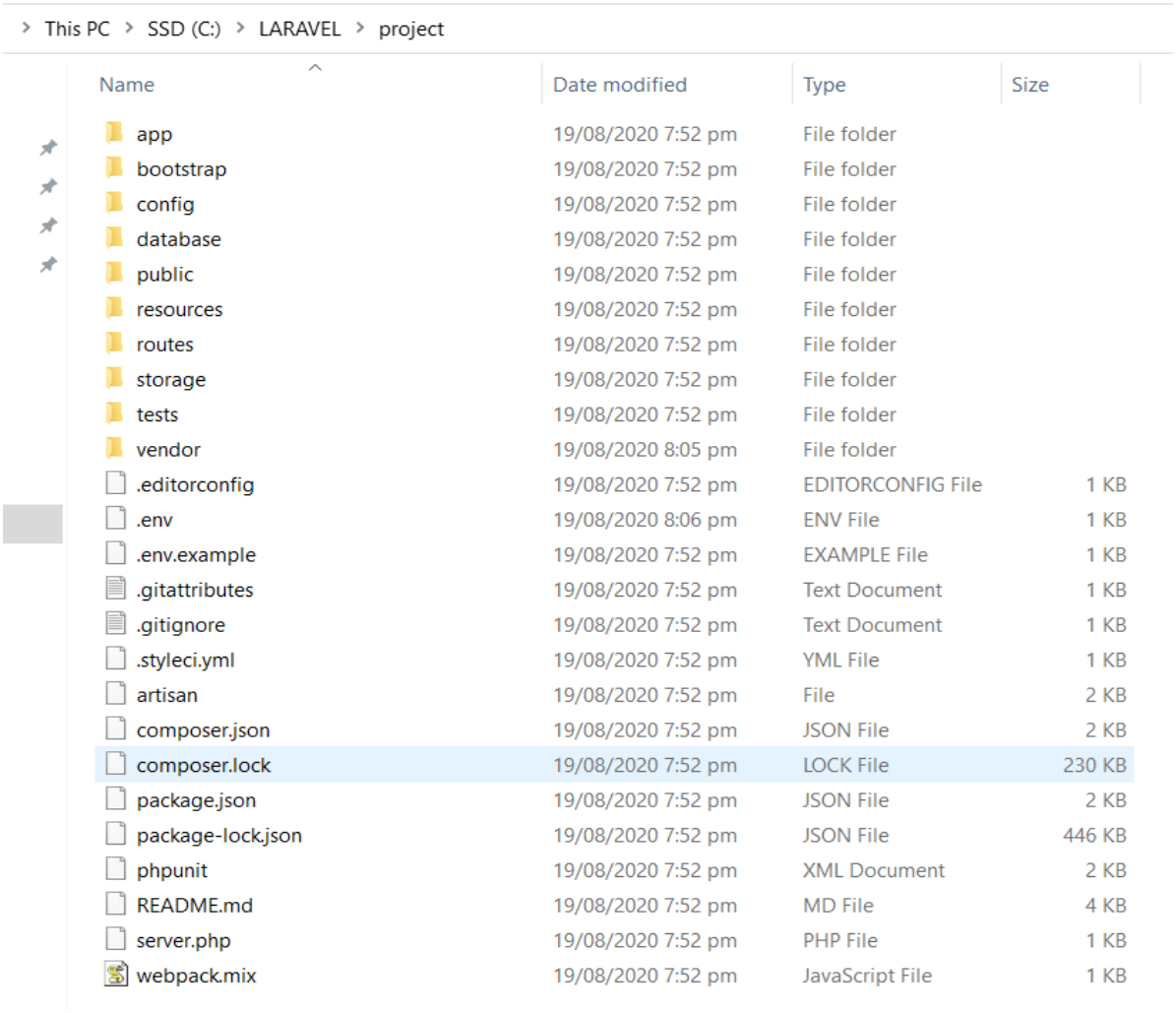
- For Laravel, PHP version >= 7.2.5
- Open Command Line Interface (cmd) and run as Administrator.
- After that run global dependencies.
  - composer global require laravel/installer.
- This command “composer global require laravel/installer ”will download and install laravel executable on your computer.

Laravel Installation

- After installation, Next step is to create new project.
  - Use command: laravel new ‘project name’.
- Next start the Laravel development server.
  - Use command: php artisan serve

Folder Structure of Laravel

- After creating a new project, you can explore it:



Folder Structure of Laravel

- **app folder:** is directory where bulk of our actual application will go, Models, controllers, route definitions and our PHP domain code all go in here.
- **bootstrap folder:** contains the files that the Laravel framework uses to boot every time it runs.
- **config folder:** where configuration of our application go.
- **databases folder:** where migration and seeds will go.
- **public folder:** where our assets go, these all files that publicly available to our application.

Folder Structure of Laravel

- **resource folder:** non-PHP files that are needed for other scripts live here. Views, language files, and (optionally) Sass/LESS and source JavaScript files live here.
- **storage folder:** where all the cache and log files are going to live. This is all builtin with laravel. E.g. error logs and access logs.
- **test folder:** it depends, if you are using any type of unit testing framework, then you can put all test inside this folder.
- **vendor folder:** where all third parties library will live, composer packages are going to be installed.

Folder Structure of Laravel

- **Composer.json:** where we define, which composer packages we want to include in our application.
- **.env:** this is where our database credentials will go and couple of other configuration.
- **project→routes→web.php** (this is where all routes for application are present)