COMP8005 – Applied Web Development

High-level concepts: Frameworks, MVC, Layered Architectures, Convention over Configuration, Object-Relational Mapping

Topics

- Definition of a framework
- Inversion of control
- MVC architectural pattern
- Layered architecture
- Object-Relational Mapping
- Convention over configuration

Software Development Frameworks

- Laravel (for PHP) is an example of a software development "framework"

  framework noun **1** a basic supporting structure. **2** a basic plan or system. **3** a structure composed of horizontal and vertical bars or shafts.

  Chambers Dictionary

In computer programming, a **software framework** is an abstraction in which software providing generic functionality can be selectively changed by user code,
thus providing application specific software. A **software framework** is a universal, reusable software platform used to develop applications, products and solutions. **Software frameworks** include support programs, compilers, code libraries, an application programming interface (API) and tool sets that bring together all the different components to enable development of a project or solution.

Wikipedia

A software framework is a set of source code or libraries which provide functionality common to a whole class of applications. While one library will usually provide one specific piece of functionality, frameworks will offer a broader range which are all often used by one type of application. **Rather than rewriting commonly used logic**, a programmer can leverage a framework which provides often used functionality, limiting the time required to build an application and reducing the possibility of introducing new bugs.

DocForge – http://docforge.com/wiki/Framework

Frameworks

- Think about all of the things the average medium to large-scale application might do...
  - Connect to a database
  - Present information to a user view
  - Log errors to a file
  - Authenticate a username and password
  - Ensure that transactions are atomic
  - Authorise a user to access a certain web page
  - Send messages to other applications via web service calls, remote procedure calls or messages to a queue
  - And the list goes on

Frameworks

- Without a framework, detailed specific knowledge would be required to implement all of the above.
- A good framework will provide the developer with APIs to simplify the implementation of such functional areas.
- For example, you may have encountered standard data access libraries such as JDBC (Java DataBase Connectivity) or PDO (PHP Data Objects).
- You saw how many steps are required to query a DB:
  - Open a connection
  - Check if the connection succeeded
  - Issue an SQL query
  - Check if the query succeeded
  - Deal with the query results
  - Close the connection

Frameworks

- You will see as part of Spring Framework (and others) that this is greatly simplified and a lot of the “plumbing” or “boilerplate” code has been hidden from the developer.
- Good frameworks will provide “abstractions” for many of those horribly difficult to use and bug-prone technologies we grudgingly develop, e.g.
  - Messaging, e.g. JMS or STOMP
  - Database Access, e.g. JDBC or PDO
  - Remoting, e.g. RMI
  - Web Services, e.g. REST or SOAP
  - Transaction Management, e.g JTA
- Spring provides Template-style classes that greatly simplify the above, as do some other frameworks

Inversion of Control

- Inversion of Control (IoC) is a term that seems to have been coined in a 1988 paper
  - http://www.laputan.org/drc/drc.html (Recommonded reading)

"One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. This **inversion of control** gives frameworks the power to serve as extensible skeletons. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application."

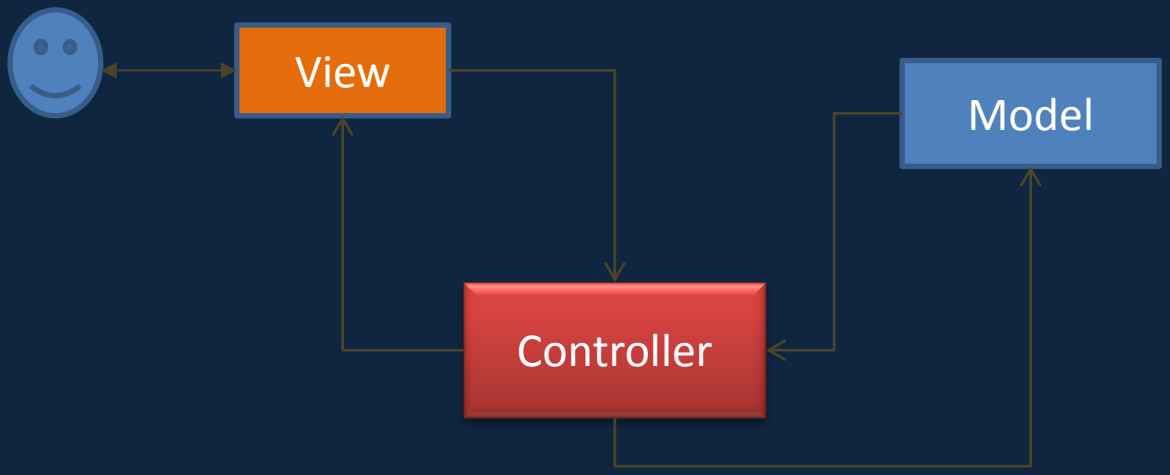
Inversion of Control

- You should read the following short blog post:
  - http://martinfowler.com/bliki/InversionOfControl.html
- It references the 1988 paper
- You will see me reference Martin Fowler several times during this module
- A type of IoC is “Dependency Injection”
- Integral to Spring (for Java) framework

Overview of MVC

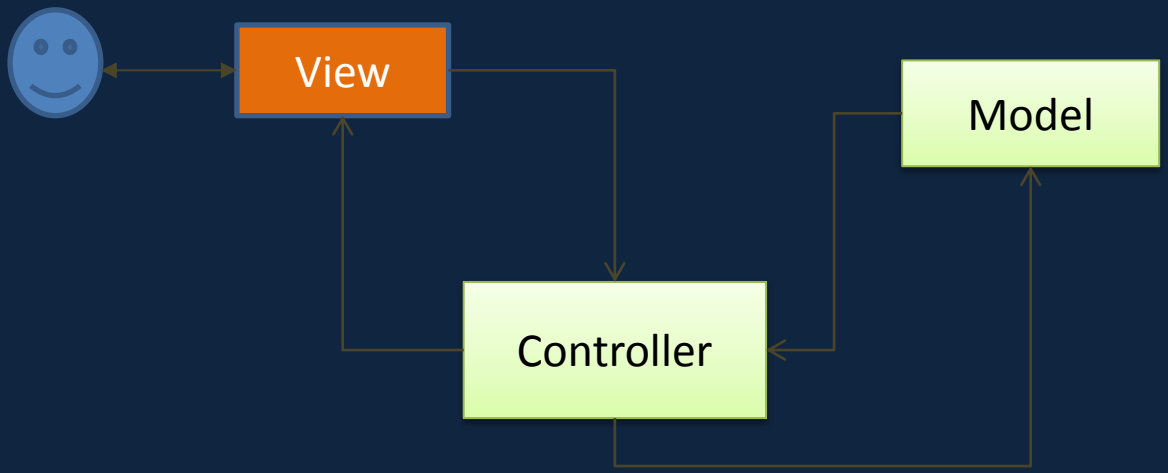
- Laravel (for PHP), which we will look at throughout this module is an **MVC** framework
- One of the Spring (for Java) projects, Spring MVC provides similar functionality
- **M**odel – **V**iew – **C**ontroller
- MVC is an "architectural style"
  - Style rather than standard because each framework will approach it slightly differently but stick to the style or spirit of the architecture

MVC Architecture



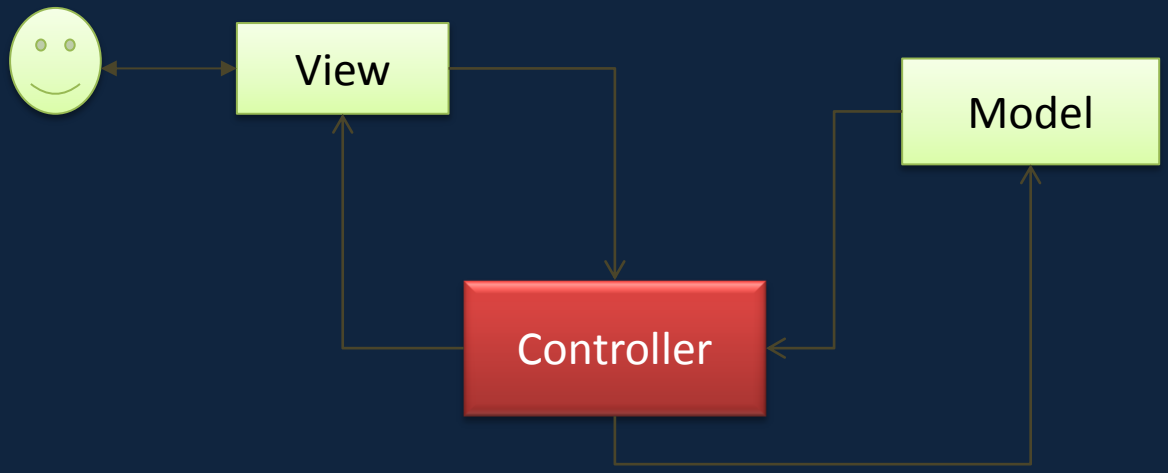
MVC Architecture

- User (or another system) interacts with **View** – possibly a HTML page - performs actions, provides data, receives page updates, etc.



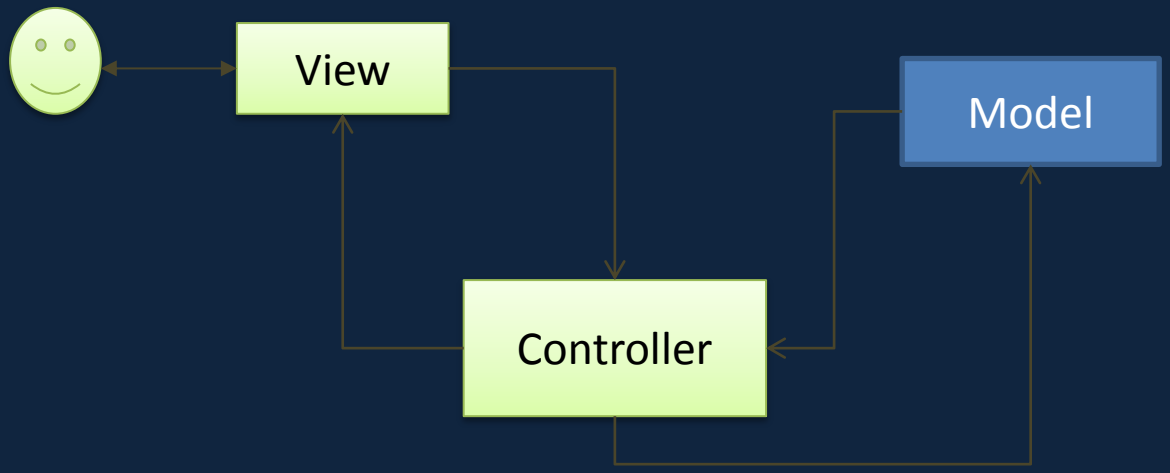
MVC Architecture

- **Controller** receives requests from the **View**
- Interacts with the **Model** to retrieve or persist data
- Sends results back to the user via a **View**



MVC Architecture

- **Model** encapsulates data and the functionality and business rules associated with the data

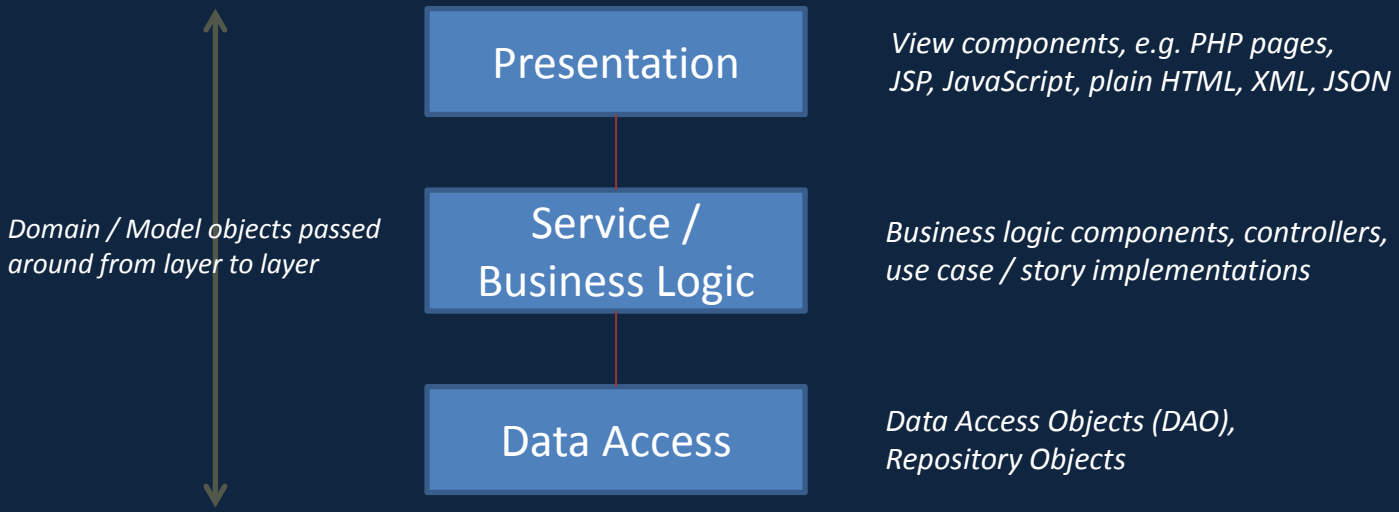


MVC Architecture

- MVC is very good from an application design and programming perspective
- It allows for a **separation of concerns**
- How to present data is a "concern"
- How to interact with data is a "concern"
- Having the MVC architecture allows us to have separate components that concentrate on presentation of data (view components), data encapsulation (model components) or routing and service layer / business logic components (controller and lower layers) – avoids "code tangling" and makes them more reusable
- Can also have additional "layers" to break things down further

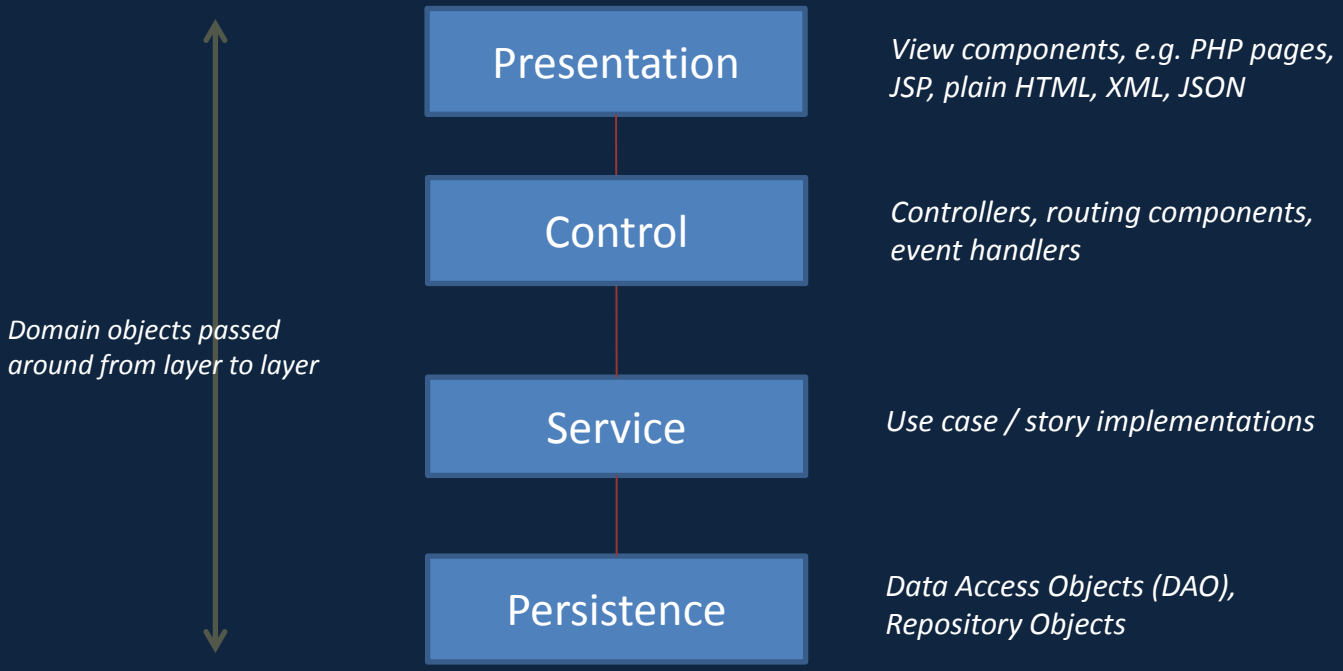
Layered Architecture

- Traditional layered architecture



Layered Architecture

- This is just one alternative – there are many others



Layering

- By layering, we make our applications more scalable and maintainable
- Looser coupling – components aren't tied to each other so can more easily reuse them or swap them in or out
- Greater cohesion – modules do one thing rather than lots of tangled-up things, which is difficult to maintain – this makes them more portable and easier to understand

The Domain

- Domain objects are the classes that represent the tangible and intangible objects in the business domain
- “Things” like Customer (tangible), Order (intangible), Product (tangible), Rental (intangible), and so on
- Decision then is where to put the domain “logic” (or business logic) – the implementation of all those user stories or use cases
- Do we encapsulate this in the domain objects or in service layer components?

Service Layer

- Most common approach nowadays is to implement the business logic or user stories / use cases in a service layer
- The domain object then is a simple object that is a container for the object's data with accessors (getters) and mutators (setters)
- The service layer object can then delegate to other objects, such as data access objects (DAOs) that interact with data stores, or web service components

Data Access Objects

- DAO is an example of a pattern, i.e. a predefined way of implementing something – think of it like a knitting pattern in a catalogue of knitting patterns
- Implement typical methods, such as *getByID*, *findAll*, *save*, to query and persist from / to database tables
- Domain object is passed into, and returned from, the DAO methods

Object-Relational Mapping

- Implementing a DAO requires manual coding to get data from a DB and put it into a domain object
- Alternative is to configure an object-relational mapper to automatically map from database tables to domain objects
- Popular example is Hibernate for Java, which implements the Data Mapper pattern
- We will also look at code samples for ActiveRecord-based ORMs

ActiveRecord Pattern

- Made popular in another framework, Ruby on Rails (we will examine that briefly much later)
- Simplest of all data access patterns
- Table maps directly to domain object
- For each table column there is a

Model Class in Laravel

```
<?php
  class Book extends Eloquent { }
?>
```

- No content?
- No requirement to write your own methods / functions
- You are inheriting some basic functions from the Eloquent ORM class

Model Class in Laravel

- e.g. *`all()`* retrieves all rows from a table
- We make a static call to `Book::all()` to retrieve all book rows
- E.g.

```
$books = Book::all();
foreach($books as $book) {
    // do something with the $book,
    echo '<p>' . $book->title . '</p>';
}
```

Convention over Configuration

- Traditionally, complex software required a lot of configuration (e.g. XML)
- By following conventions, configuration can be reduced or even eliminated
- E.g. a folder structure where there is a place for everything and everything in its place

Convention over Configuration

- A new Laravel application will have app, controller, views and models folders
  - No need for a config file to tell the app where the controllers are because they can be located by convention
- Also, Model class names match table names, but with the class singular and the table name plural, e.g. class Book, table books

Convention over Configuration

- Can use Maven "archetypes" to create projects with pre-set folder structures, e.g. src/main/java, src/test/java, etc.
- In Spring, can reference some beans / class instances even if we haven't configured them **if** we use a default bean reference, e.g. The *@Transactional* annotation assumes the existence of a *transactionManager* bean if you omit the value parameter

Summary

- We have covered some of the concepts that underpin Spring (for Java), Laravel (for PHP), and others
  - Frameworks
  - Inversion of Control
  - MVC architectural pattern
  - Layering
  - Object-relational mapping
  - Convention over configuration