FRESNO STATE
Discovery. Diversity. Distinction.

Web Programming (CSci 130)

Department of Computer Science
College of Science and Mathematics
California State University Fresno
H. Cecotti

Learning outcomes

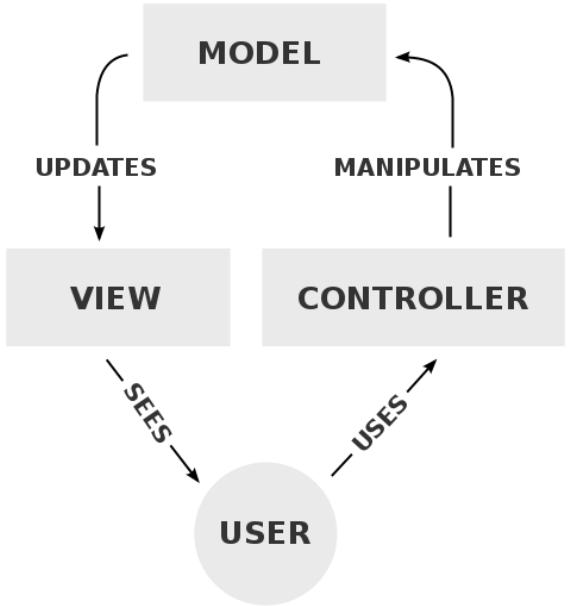
- Architectural patterns
  - **Model View Controller** (MVC)
  - Active record pattern
- What are the current main PHP frameworks
  - MVC PHP web application framework
- PHP Frameworks
  - Focus on Lavarel

Rationale

- **PHP**
  - Most popular sever side scripting language
- Too complicated to start from scratch
  - Need of **libraries** and **frameworks** to develop fast and efficiently
- Need of robust software programming design principles and practice
  - For large projects
  - **Architectural patterns**
    - → For good practice
  - When you understand how your development tools function, you feel more comfortable and confident using them.
- In order to appreciate the features
  - Program with raw PHP ☺ … CSci130 Project

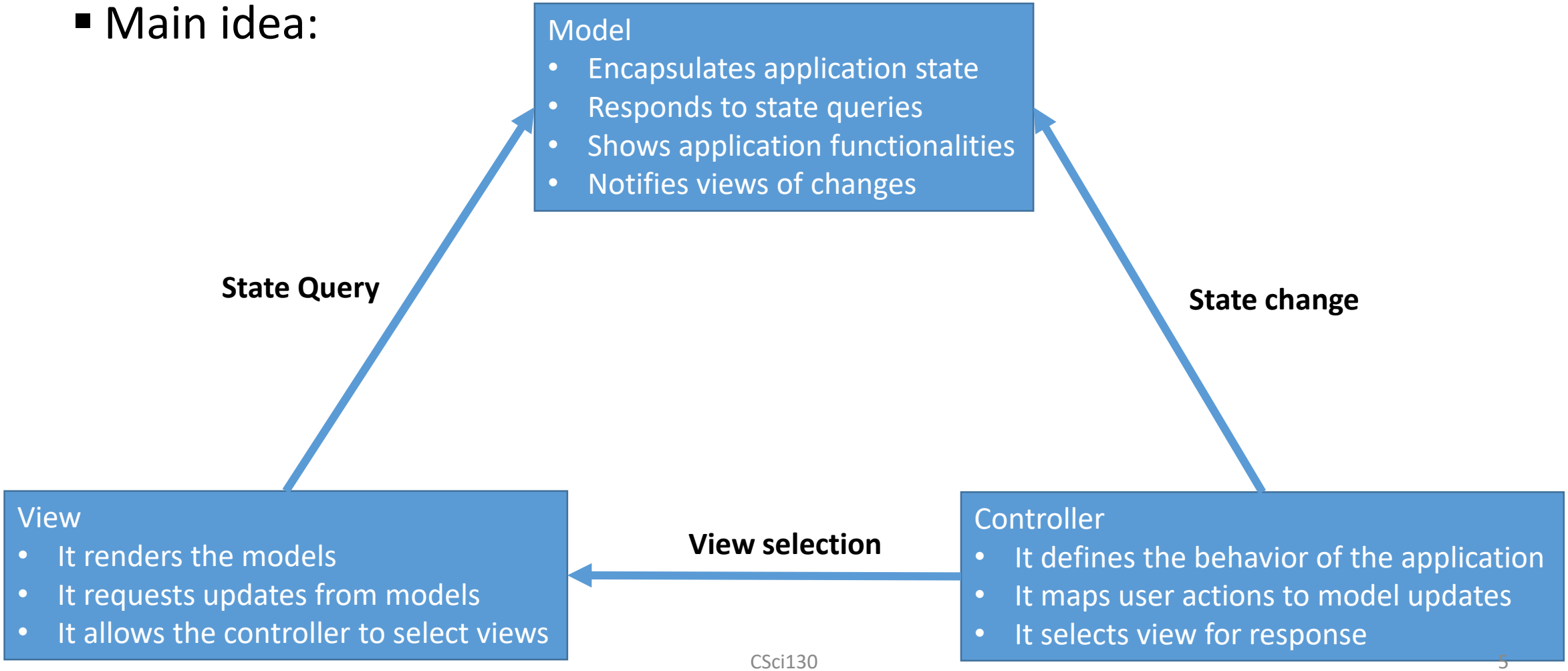
Model View Controller

- Model–view–controller (**MVC**)
  - Software architectural pattern for implementing UIs on computers
    - For interactive applications
  - Separation between
    1. The internal representations of information
    2. The ways information is presented to/accepted from the user
- Implementation (1 class/component)
  - **Model**
    - The actual internal representation
  - **View**
    - A way of looking at or displaying the model
  - **Controller**
    - It provides for user input and modification



MVC

- Main idea:



MVC

- Main concept: to decouple the 3 different elements:
  1. the **Model**
     - It handles both the behavior and data of the application domain
       - requests from users to just read the data → handled from the view
       - update of the data → handled by the controller
     - Component interacting with the database
  2. the **View**
     - The UI and presentation
  3. the **Controller**
     - It is handling the interaction between View and Model, controlling the flow

MVC

- Model (**how to do**)
  - Element doing the "work"
    - It models the real problem being solved
    - It deals with data (e.g. data storage and retrieval)
  - It should be independent of both the **Controller** and the **View**
  - It provides services for them to use
  - It can have more than one view
  - It is **not** the Database → combination of Data and Business Logic required to perform the actions on the DB
  - Advantages
    - Independence → flexibility and robustness

MVC

- Controller (**what to do**)
  - ➢ It tells what the model has to do
    - e.g. the user can control thanks to the GUI
      - → GUI == Controller
  - ➢ Separation between the Controller and the Model
  - ➢ The design of the Controller depends on the Model
    - Controller design = f(Model) : TRUE
  - ➢ The Model should **not** depend on the Controller
    - Model = f(Controller) : FALSE

MVC

- View (**what is being done**)
  - It shows what the model is doing
    - Passive observer
    - No impact on the model
  - The user should see what the application is doing
  - It is responsible for displaying the information in a way that the user can understand
  - The Model should be independent of the View
  - The View should ***not*** display what the Controller ***thinks*** is happening

MVC

- In relation to the GUI,
  - **Views:** components that display the application's user **interface** (UI)
    - only **display**s information
    - UI: created from the model data
    - Example:
      - Creation of the table that displays the result of a query
  - **Controllers:** components that handle user **interaction**
    - To work with the model + to select a view to render that displays UI
    - It handles and responds to user **input** and **interaction**.
    - Example:
      - The controller handles query-string values, it passes them to the model, the model may use these values to query the database

MVC

- To **never** mix Model code with Controller (GUI code)
- The Model should not be contaminated with
  - ➢control code (what to do) or display code (how the result is presented)
- The View should represent exactly the Model
- The Controller should
  1. communicate *(access methods) with* the Model and the View
  2. not manipulate (no modifier methods) the Model and the View
     - i.e. the Controller can set variables that the Model and View can read
- Small project
  - ➢Combination of the view and the model

MVC

- Frameworks → choice of the framework: Client/Server → MFC ?
- Web MVC frameworks solutions
  - Thin client approach (emphasis on the server)
    - Almost the entire Model, View, Controller logic on the server.
      - e.g. Django, Rails, ASP.NET MVC
    - The client sends information to the Controller
      - hyperlink requests
      - form submissions to the controller
    - The client receives a complete and updated web page (or other document) from the View
    - The model exists entirely on the server.
  - More work for the client
    - MVC components to execute partly on the client
      - e.g. AngularJS, EmbersJS, JavaScriptMVC
- Separation of the code
  - Judicious choice of the framework and possibility to change frameworks/technology over time
  - /!\
    - Maintenance of the project
    - Addition of new functionalities

Object Relational Mapping (ORM)

- Goal
  - Conversion of data between incompatible type systems using OOP languages
  - **To reduce the amount of code that must be written**
- Idea
  - From database to a virtual object database
    - **Now**
      - MySQL query → Table → Array of Objects
    - **With ORM**
      - Query → Array of Objects
  - Document oriented database (direct)
    - JSON → Array of Objects
    - XML → Array of Objects
- Target
  - List<Student> ls = Student.Get(Student.Properties.Grade == 'A');

ORM

- PHP Active Record
  - http://www.phpactiverecord.org/
  - See file phpactiverecord.php (examples related to CRUD functions)
- Eloquent ORM
  - Advanced PHP implementation of the ARP
    - + internal methods for enforcing constraints on the relationships between database objects
  - Presentation of database tables as classes
    - with 1 object instance ← → 1 raw in the table

Active Record Pattern

- Stores in-memory object data in RDBMs
  - The interface of an object conforming to this ARB
    - **Functions**: Insert, Update, and Delete
    - **Properties**: the columns in the underlying DB table
- We started this approach ☺ ! (see rdbms.php file)
- ARP
  - An approach to accessing data in a DB
  - A DB table or view is wrapped into a class
    - →**instance** is tied to **a single row** in the table.
  - After the creation of an object → a new row is added to the table
  - Any object loaded gets its information from the DB
  - When an object is updated, the corresponding row in the table is also updated !!
    - The wrapper class implements:
      - Accessor methods
      - Properties for each column in the table or view.

Migration

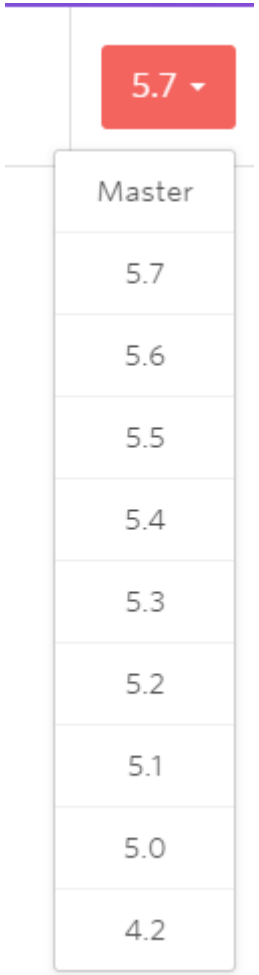
- **Migration**: moving from one platform to another
  - Classes to Database / Database to Classes
- Database independent applications
  - Aim at the lowest common denominator of all of your database platforms
    - → lowering the bar on the high-performance features that can be used on the platform you are currently using.
- Version control for your DB
  - To allow your team to easily modify and share the application's database schema.
  - Paired with Laravel's schema builder to easily build your application's database schema.
    - E.g. migration to solve the issue of:
      - tell a member of the project to manually add a column to their local DB schema

Frameworks

- Laravel (2011)
  - https://laravel.com/
  - https://laravel.com/docs/5.5
- Symphony (2005)
  - https://symfony.com/
- Zend (2006)
  - http://www.zend.com/
- CodeIgniter (2006)
  - https://codeigniter.com/
- Yii 2 (2008)
  - http://www.yiiframework.com/
- CakePHP (2005)
  - https://cakephp.org/

Laravel

- Released in 2011
  - Current version 5.7
    - https://laravel.com/docs/5.7
- Install
  - **Step 1**: Install composer
    - Dependency Manager for PHP
      - https://getcomposer.org/download/
  - **Step 2**:
    - composer global require "laravel/installer=~1.1“
    - lavarel new mydir
- Creation of a project
  - Model: Database
  - Controller
  - Routes (different versions, different ways to do it ☹)
    - Definition of all the routes related to our resource(s) (Tables)
    - POST/GET/DELETE …
  - View:



Some features

- **Artisan**
  - the command-line interface included with Laravel.
- **Blade**
  - template engine
  - No restriction from using plain PHP code in the views.
    - all Blade views are compiled into plain PHP code
      - cached until there is a modification
  - @ directives
    - Controls, loops, …
  - {{ $car->make }} → <?php echo $car->make; ?>
- Eloquent ORM
  - simple ActiveRecord implementation for working with your database
- See files
  - php_activerecordexample.php

Conclusion

- MVC
  - Controllers
    - to handle user requests and retrieve data, by leveraging Models
  - Models
    - to interact with your database and retrieve your objects' information
  - Views
    - to render pages
- Time
  - MVC (70s) , Active Record Patterns (2003), PHP (1995), … → Laravel (**2011**)
- For job interviews
  - Remember the key frameworks, key functionalities
  - Modify your existing projects to take advantage of these frameworks
    - Angular, Node, Laravel, …
- Further readings
  - See links on Canvas