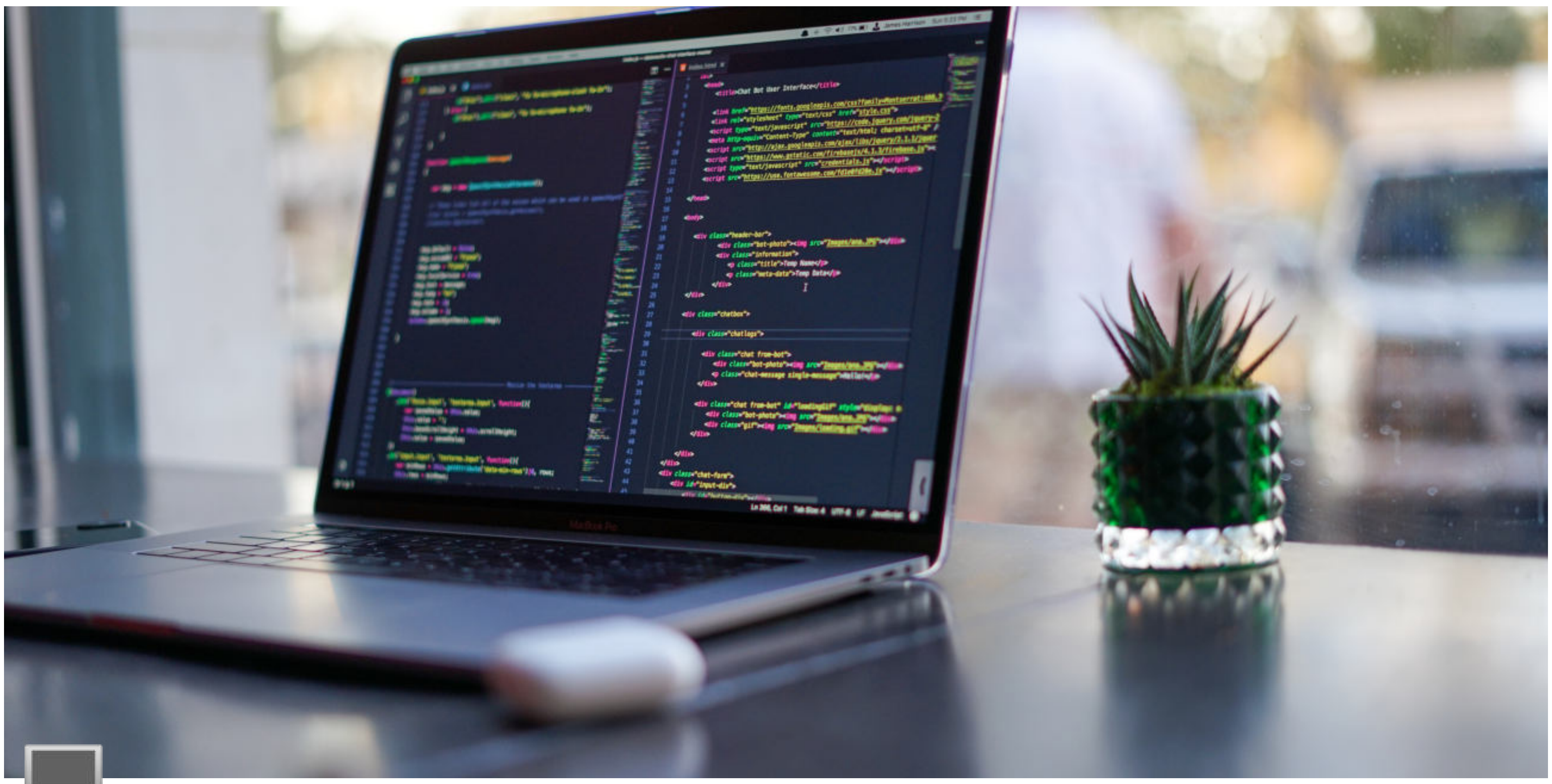


💻

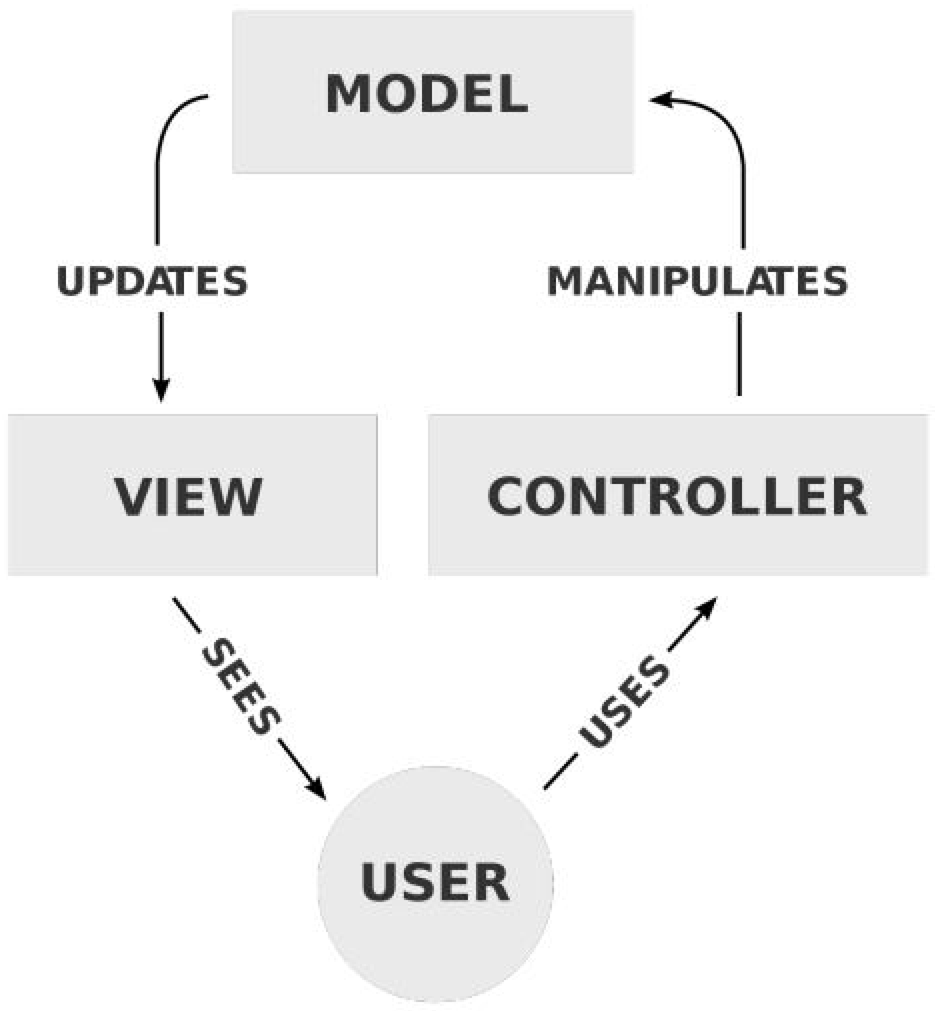
Week 5 Lecture 1

Class	BSCCS2003
Created	@October 4, 2021 12:18 AM
Materials	
Module #	27
Type	Lecture
Week #	5

MVC Origins

Model-View-Controller

- Design pattern: or collection of design patterns
- Originally introduced in the context of GUI design in Smalltalk-80
- Many different variants, interpretations, ...



By RegisFrey - Own work, Public Domain, Wikipedia

From: Trygve Reenskaug

Date: 10 December 1979

MODELS-VIEWS-CONTROLLERS

MODELS

Models represent knowledge. A model could be a single object (rather uninteresting), or it could be some structure of objects. ~~The proposed implementation supports knowledge represented in something resembling *semantic nets* (If I understand Laura correctly)~~

VIEWS

A view is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a *presentation filter*.

A view is attached to its model (or model part) and gets the data necessary for the presentation from the model by asking questions. It may also update the model by sending appropriate messages. All these questions and messages have to be in the terminology of the model, the view will therefore have to know the semantics of the attributes of the model it represents. (It may, for example, ask for the model's identifier and expect an instance of Text, it may not assume that the model is of class Text)

CONTROLLERS

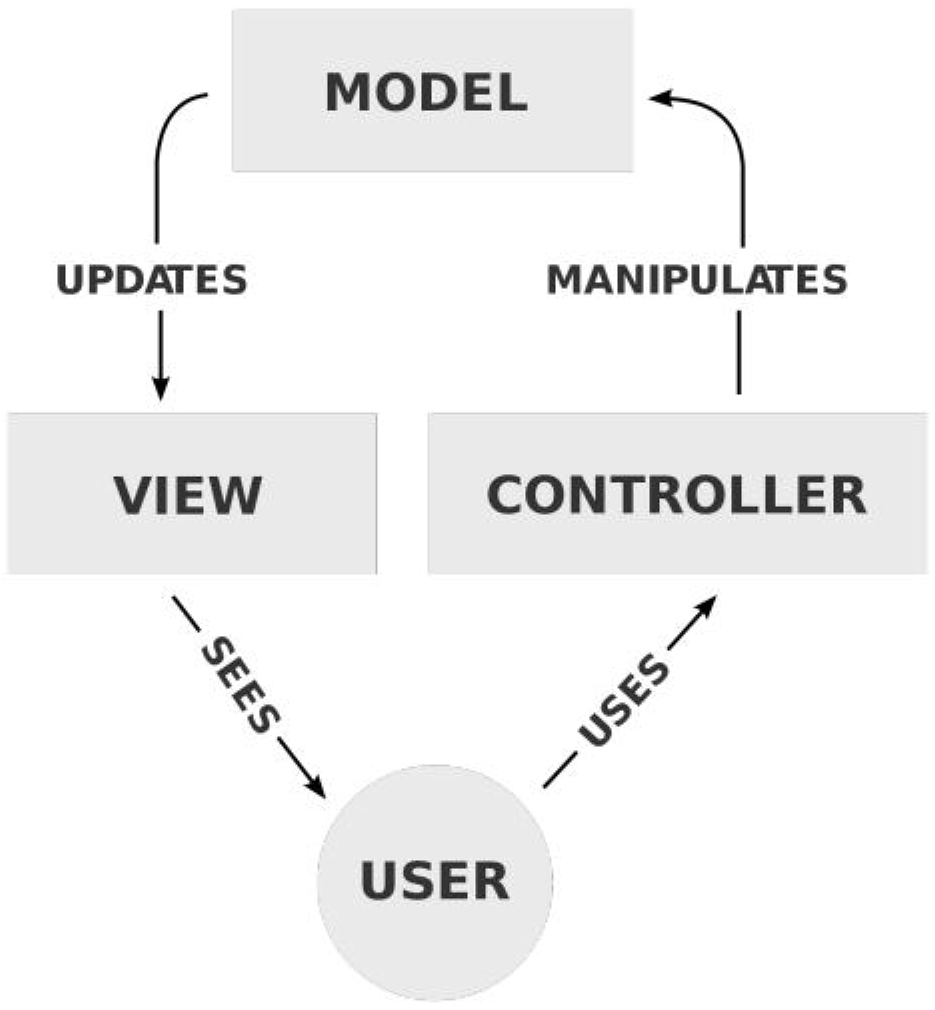
A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and pass these messages on .to one or more of the views.

A controller should never supplement the views, it should for example never connect the views of nodes by drawing arrows between them.

Conversely, a view should never know about user input, such as mouse operations and keystrokes. It should always be possible to write a method in a controller that sends messages to views which exactly reproduce any sequence of user commands.

General Concept - Action

- Take action in response to user input
- Communicate with the model, extract the view



Applicability

- Originally designed for GUI applications
- Separation of concerns - model vs views - and connection through controller
- State of interaction maintained as part of overall system memory

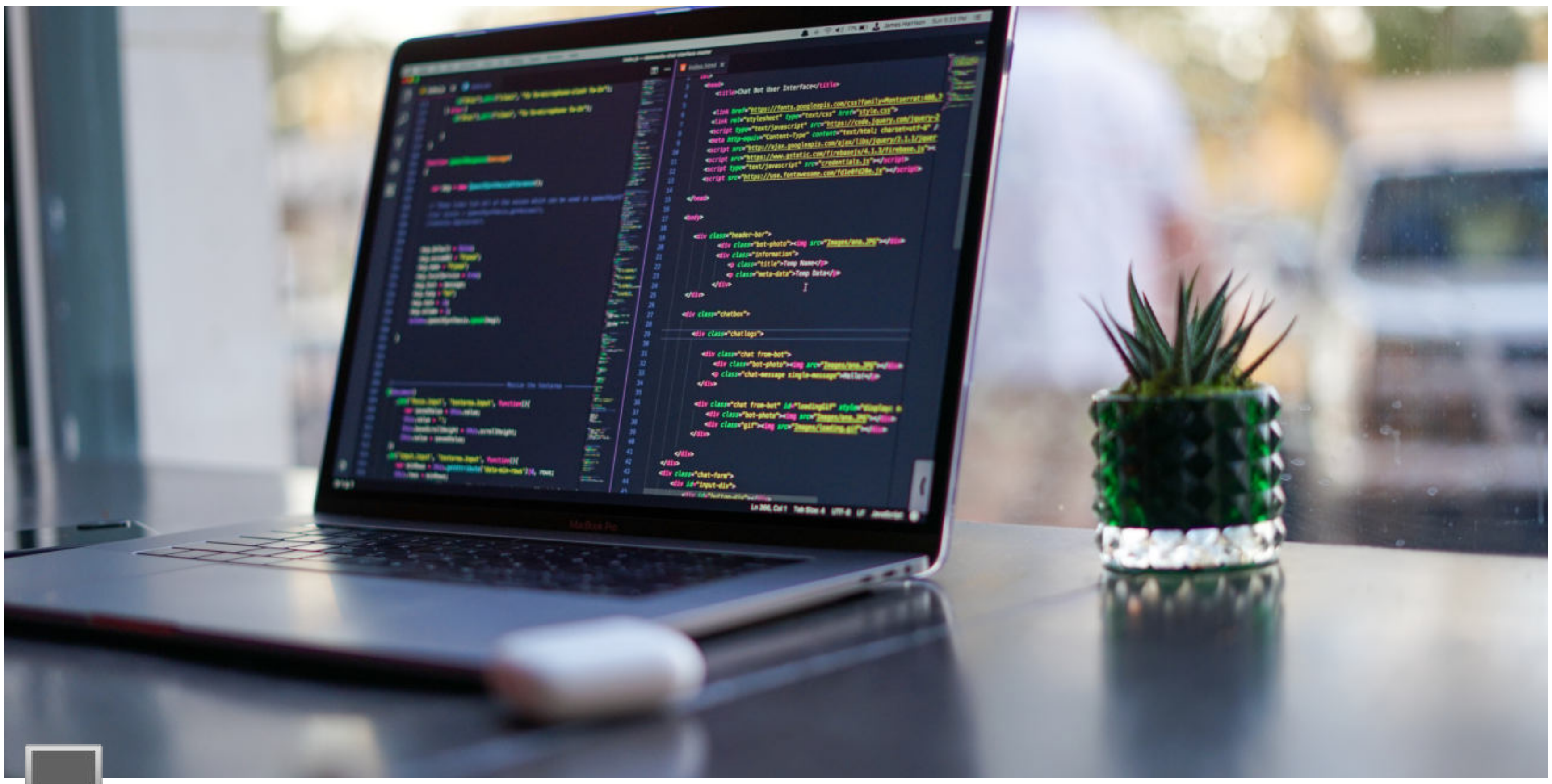
Web?

- Server does not maintain the state of the client
- Client is pure front-end to the user
- Some of the analogies break down - hence many variants of MVC

Present Context

- MVC is a good conceptual framework to understand separation of concerns
- Breaks down if applied too rigidly
- The web in general does not have the close knit structure of GUI applications needed for MVC
- Other aspects of static typing and type inference of objects also broken in Python-like languages

Apply basic learnings from MVC, but be prepared to stretch



Week 5 Lecture 2

Class	BSCCS2003
Created	@October 4, 2021 12:41 AM
Materials	
Module #	28
Type	Lecture
Week #	5

Requests and Responses

Example dynamic web page

- View: page
- Links: clickable to select various options
- Clicking a link triggers different behaviors

Request - Response

- Web is based completely on requests and responses
  - Client makes requests
  - Server sends responses
- Basic requests: clicking on link / URL
  - HTTP GET
- More complex requests: form submissions
  - HTTP POST

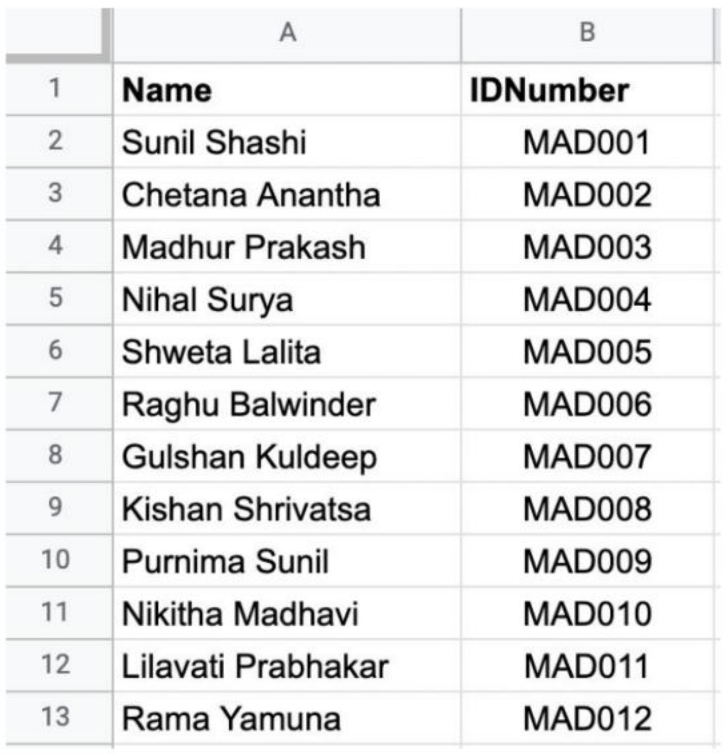
Constraints?

- Any "page" can be requested
- Assignments, quizzes, lectures, general information

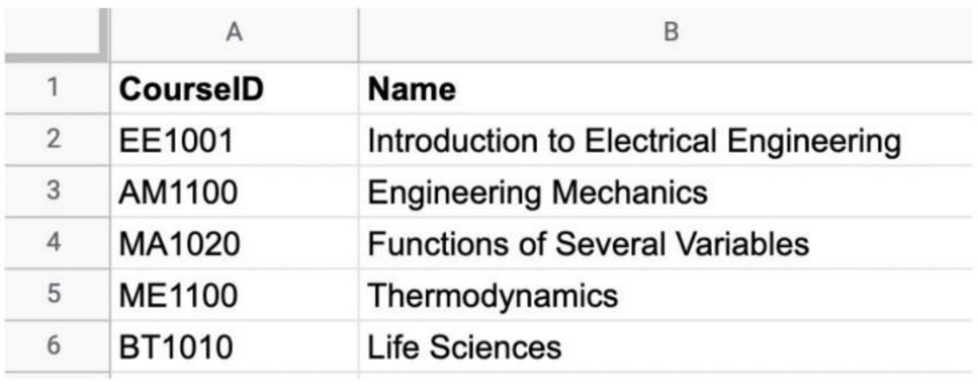
  Are there common threads?

Example: Gradebook

- Students: ID, name, address, ...
- Courses: ID, name, department, year, ...
- StudentCourse Relationship: which students are registered for which courses



	A	B
1	**Name**	**IDNumber**
2	Sunil Shashi	MAD001
3	Chetana Anantha	MAD002
4	Madhur Prakash	MAD003
5	Nihal Surya	MAD004
6	Shweta Lalita	MAD005
7	Raghu Balwinder	MAD006
8	Gulshan Kuldeep	MAD007
9	Kishan Shrivatsa	MAD008
10	Purnima Sunil	MAD009
11	Nikitha Madhavi	MAD010
12	Lilavati Prabhakar	MAD011
13	Rama Yamuna	MAD012

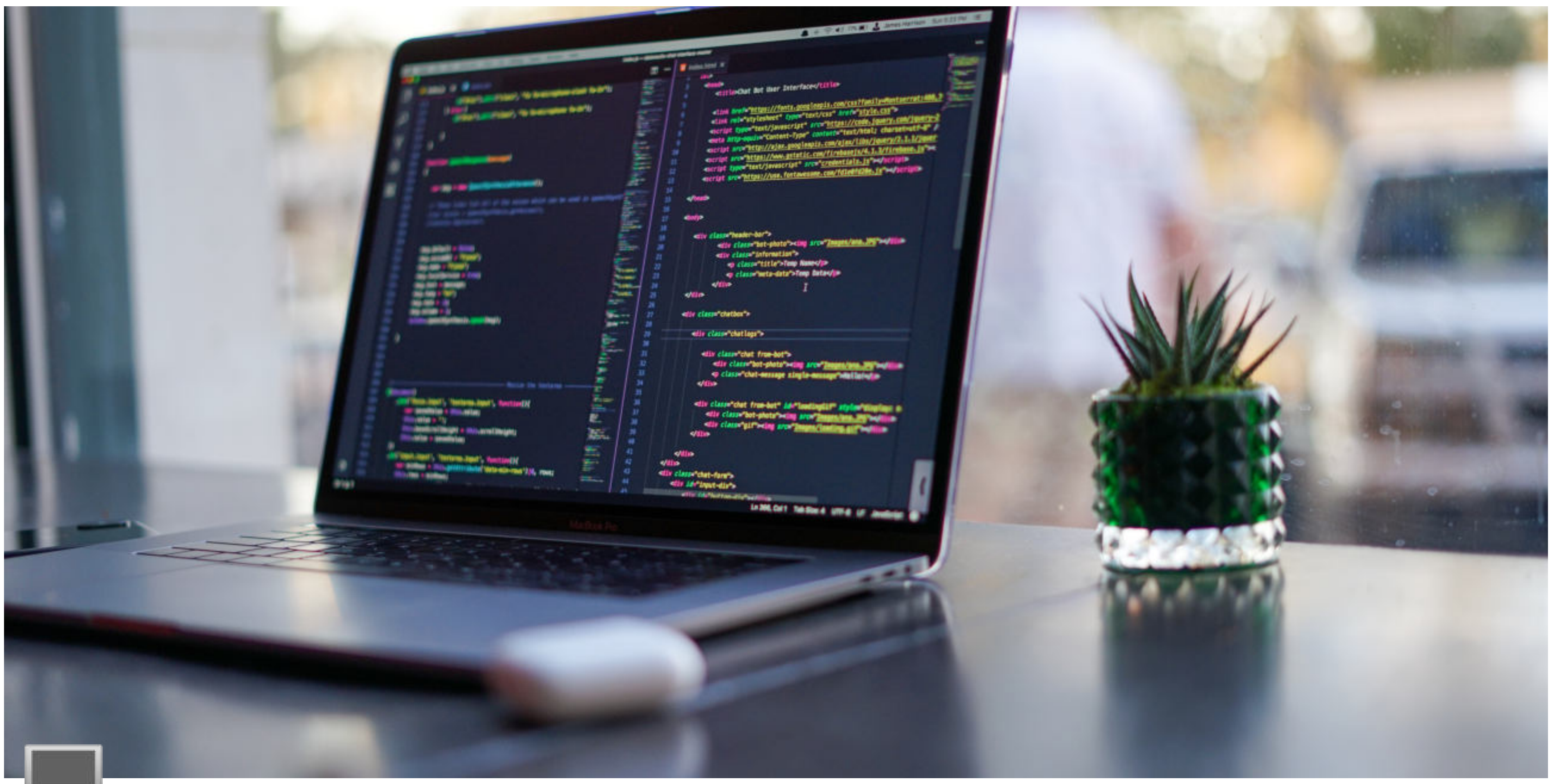


	A	B
1	**CourseID**	**Name**
2	EE1001	Introduction to Electrical Engineering
3	AM1100	Engineering Mechanics
4	MA1020	Functions of Several Variables
5	ME1100	Thermodynamics
6	BT1010	Life Sciences

Common Operations

- Create a new student - add name, roll number, date of birth, ...
- Create a new course
- Assign a student to course
- Enter marks of student / Update marks of a student
- View summaries / charts / histograms
- Archive an old course
- Remove graduated students

Some essential functions can be distilled



Week 5 Lecture 3

Class	BSCCS2003
Created	@October 4, 2021 12:51 AM
Materials	
Module #	29
Type	Lecture
Week #	5

CRUD

Types of operations - Create

- Create a new entry
- Must not already exist
- Check within database to avoid conflicts
- Mention mandatory vs optional fields (name, address, mobile number) ...

Types of operations - Read

- Get a list of students
- Summarize number of students, age distribution, geographic locations
- Plot histograms of marks

Types of operations - Update

- Change of address
- Update marks
- Change start date of course

Type of operations - Delete

- Remove graduated students
- Delete mistaken entries

- Un-enroll students from course

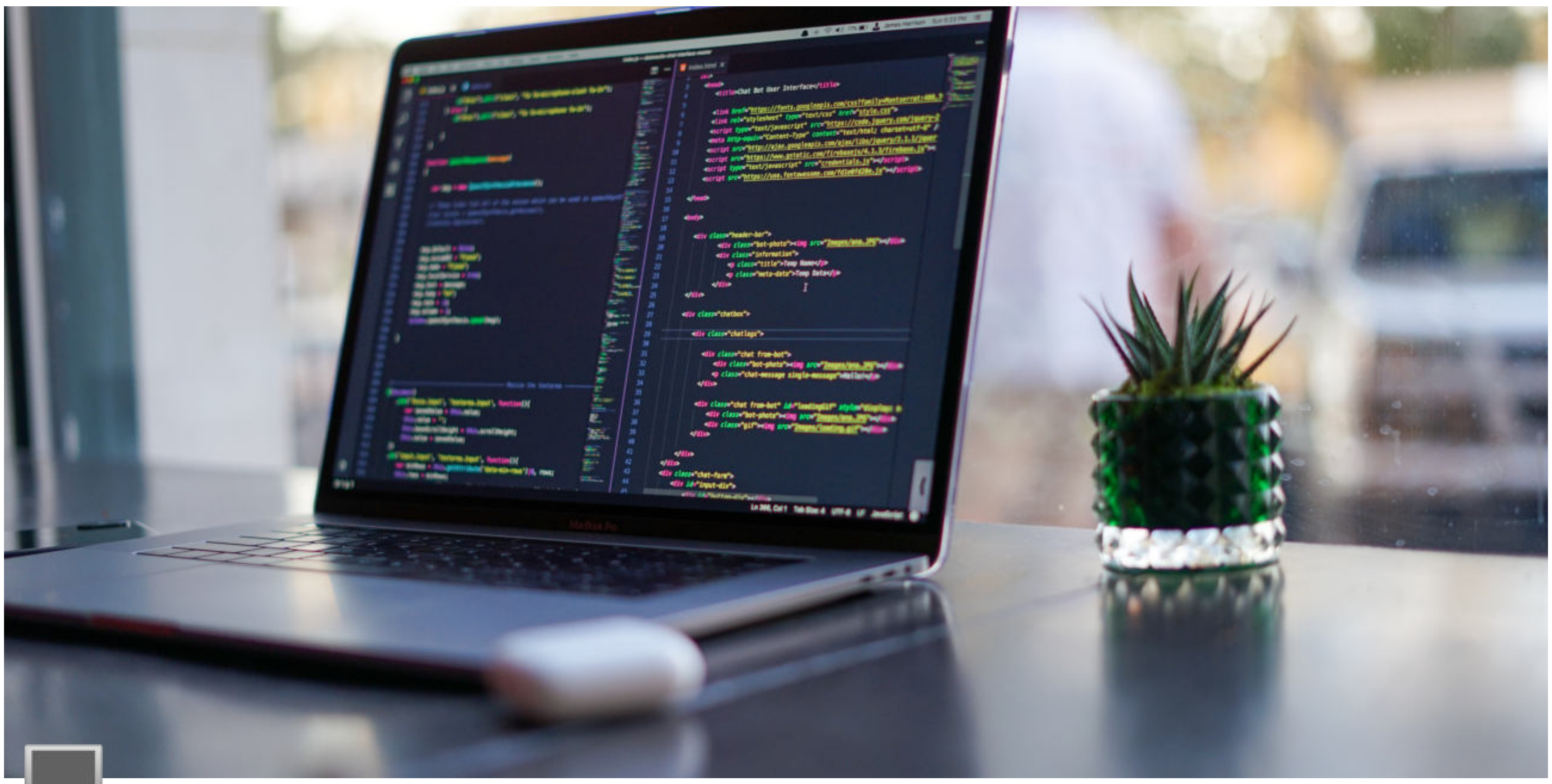
CRUD

Create - Read - Update - Delete

- Originally in context of database operations: nothing to do with the Web
- Reflects cycle of data models
- Databases optimized for various combinations of operations
    - Read-heavy: lots of reading, very little writing or creating
    - Write-heavy: security archive logs

API - Application Programming Interface

- Standardized way to communicate with a server
- Client only needs to know the API - now how the server implements the database, for example
- CRUD is good set of functionality for a basic API
    - Usually considered the first level API to implement a web application
- Deals only with the data model life cycle - other control aspects possible



💻

Week 5 Lecture 4

Class	BSCCS2003
Created	@October 4, 2021 12:59 AM
Materials	
Module #	30
Type	Lecture
Week #	5

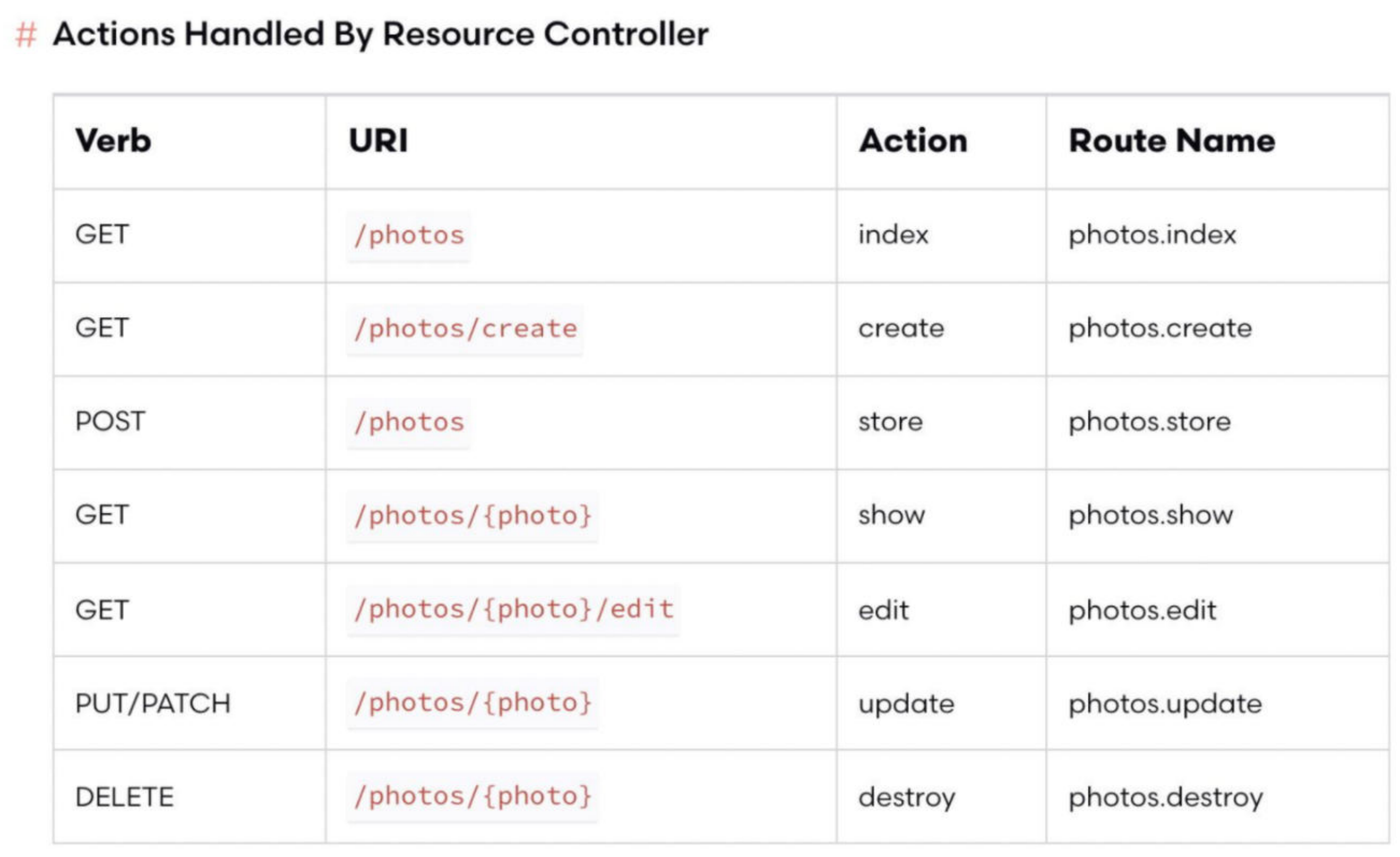
Group Actions by Controller

Actions vs Controllers?

- CRUD etc. are a set of actions
- Other actions:
    - send email
    - update logs
    - send alert on WhatsApp / Telegram
- Can actions be grouped together logically?

  Controller!

Example: Laravel PHP framework



Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Summary

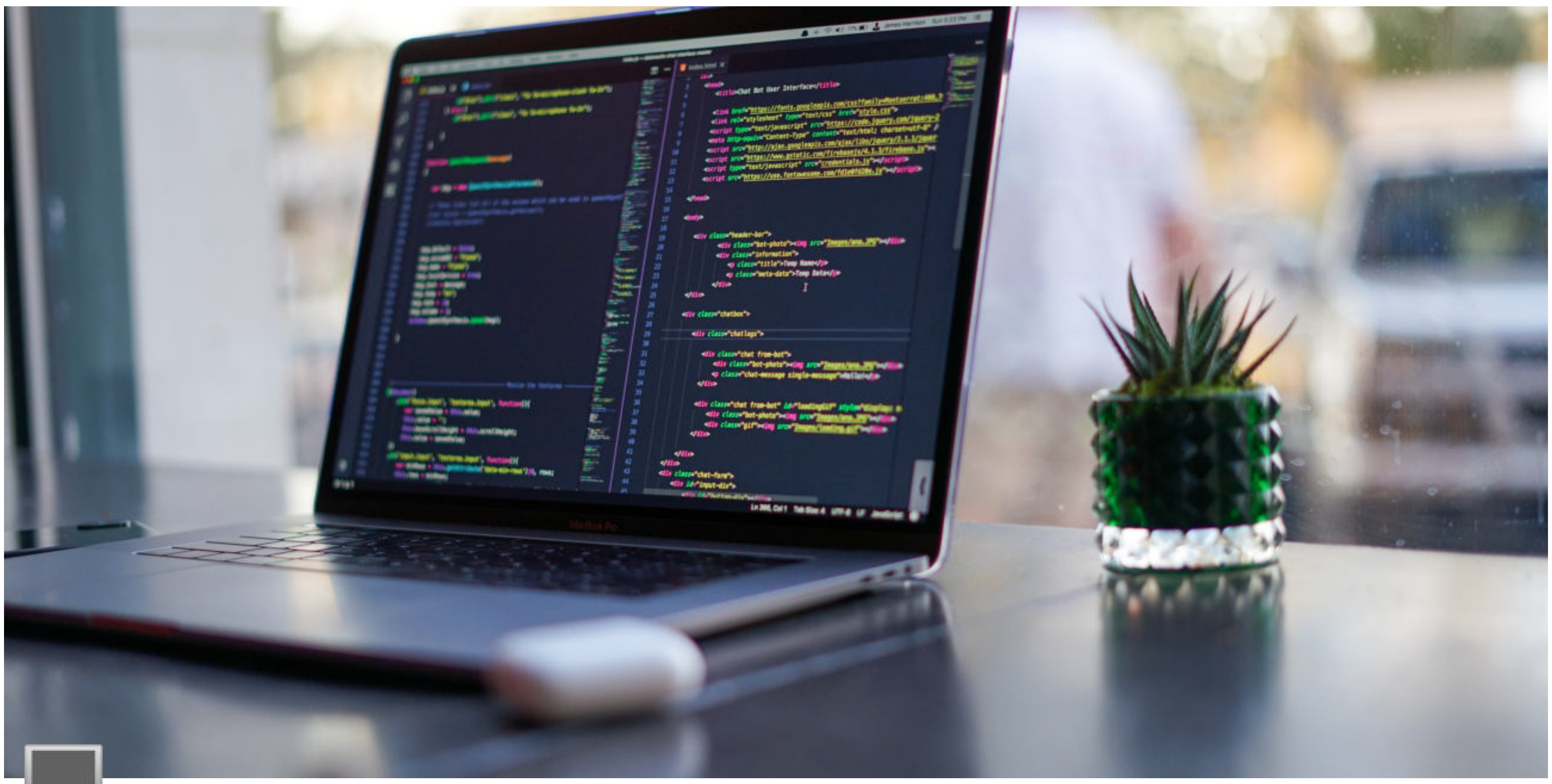
- Actions: Interaction between view and model
- Controller: Group actions together logically
- API: Complex set of capabilities of server
- Interaction through HTTP requests
- HTTP verbs used to convey meanings

Rules of Thumb

- Should be possible to change views without the model ever knowing
- Should be possible to change the underlying storage of model without views ever knowing
- Controllers / Actions should generally NEVER talk to a database directly

In practice:

- Views and controllers tend to be more closely inter-linked than with Models
    - More about a way of thinking that a specific rule of design



💻

Week 5 Lecture 5

Class	BSCCS2003
Created	@October 4, 2021 1:13 AM
Materials	
Module #	31
Type	Lecture
Week #	5

Routes and Controllers

Web applications

- Client-Server model
- Stateless: Server does not know the present state of the client
    - Must be ready to respond to whatever the client requests without assuming anything about the client
- Requests sent through HTTP protocol
    - Use variants of the GET, POST (verbs) to convey meaning
    - Use URL (Uniform Resource Locator) structure to convey context

Routing: mapping URLs to actions

Python Decorators

- Add extra functionality on top of a function
- "@" - decorators before function name
- Effectively, it is a function of a function
    - Take the new inner function as an argument
    - Return a function that does something before calling the inner function

Basic routing in Flask

```
from flask import Flask
app = Flask(__name__)
```

```
@app.route('/')
def home():
  return 'Hello, World!'
```

HTTP verbs

```
@app.route('/'. methods=['GET'])
def index():
  ...

@ppa.route('/create', methods=['POST'])
def store():
  ...
```

CRUD-like functionality

```
# Assume 'index', 'store' etc. are functions

@app.route('/', method=['GET'])(index)
@app.route('/create', method=['POST'])(store)
@app.route('/<int:user_id>', method=['GET'])(show)
@app.route('/<int:user_id>/edit', method=['POST'])(update)
@app.route('/<int:user_id>', method=['DELETE'])(destroy)
```

Summary

- Flask is not natively MVC
    - But MVC is more a way of thought than a framework
- Simple URL routing
- Helpers to do large scale routing of common functions

Structure the application with separation of concerns in mind

- MVC is just one way to achieve clean design