Model View Controller

Model-View-Controller

- MVC is one of the most widely used patterns in architecture today
- You will see it everywhere
- It’s worth paying attention to this lecture, even if your own personal design pattern is to sleep through them ;)

MVC: Model

- The model contains and manages the data and state of an application.
    - Data Schemas
    - Current State of Data
    - Data validation logic
    - Data update logic

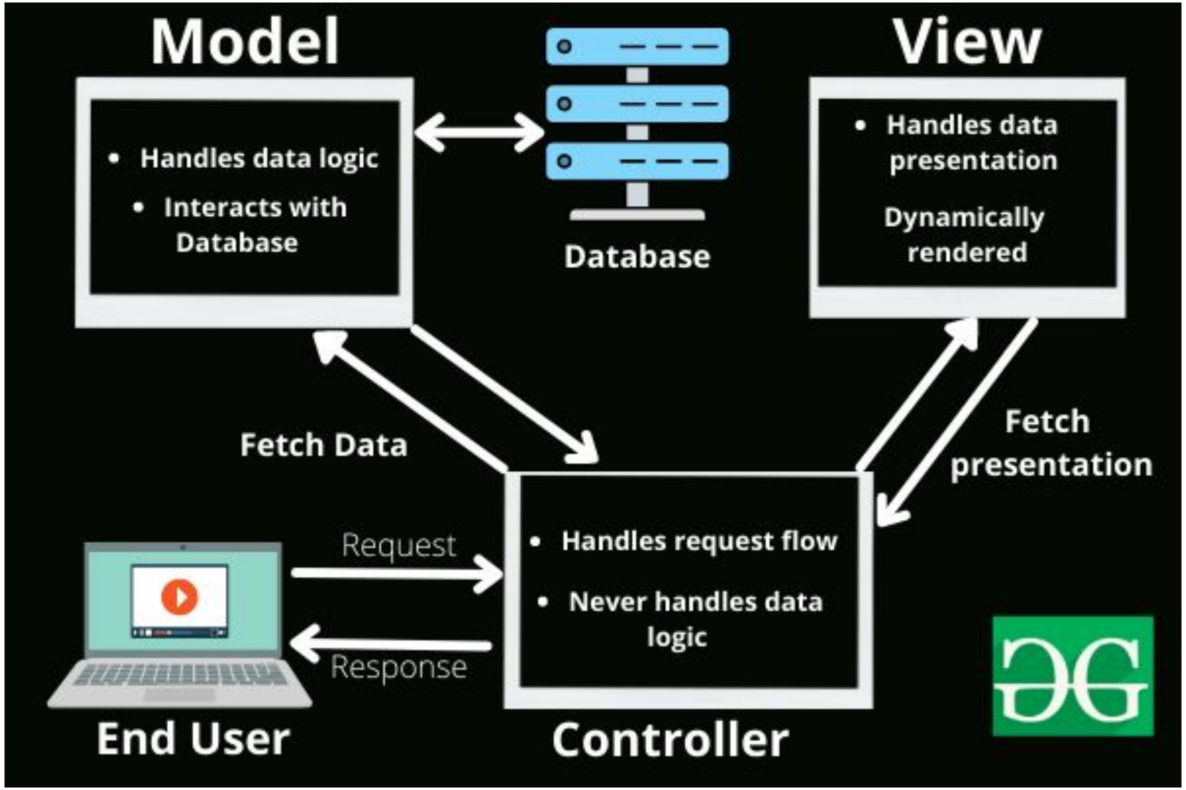
MVC: View

- The view is a visual representation of the model.
- Examples of views are:
  - a computer GUI
  - text output of a computer terminal
  - a smartphone's application GUI
  - a PDF document
  - a pie chart
- The view only **displays** the data; it doesn't **handle** it.
- Code includes UI components such as text boxes, dropdowns, and other things that the user interacts with.

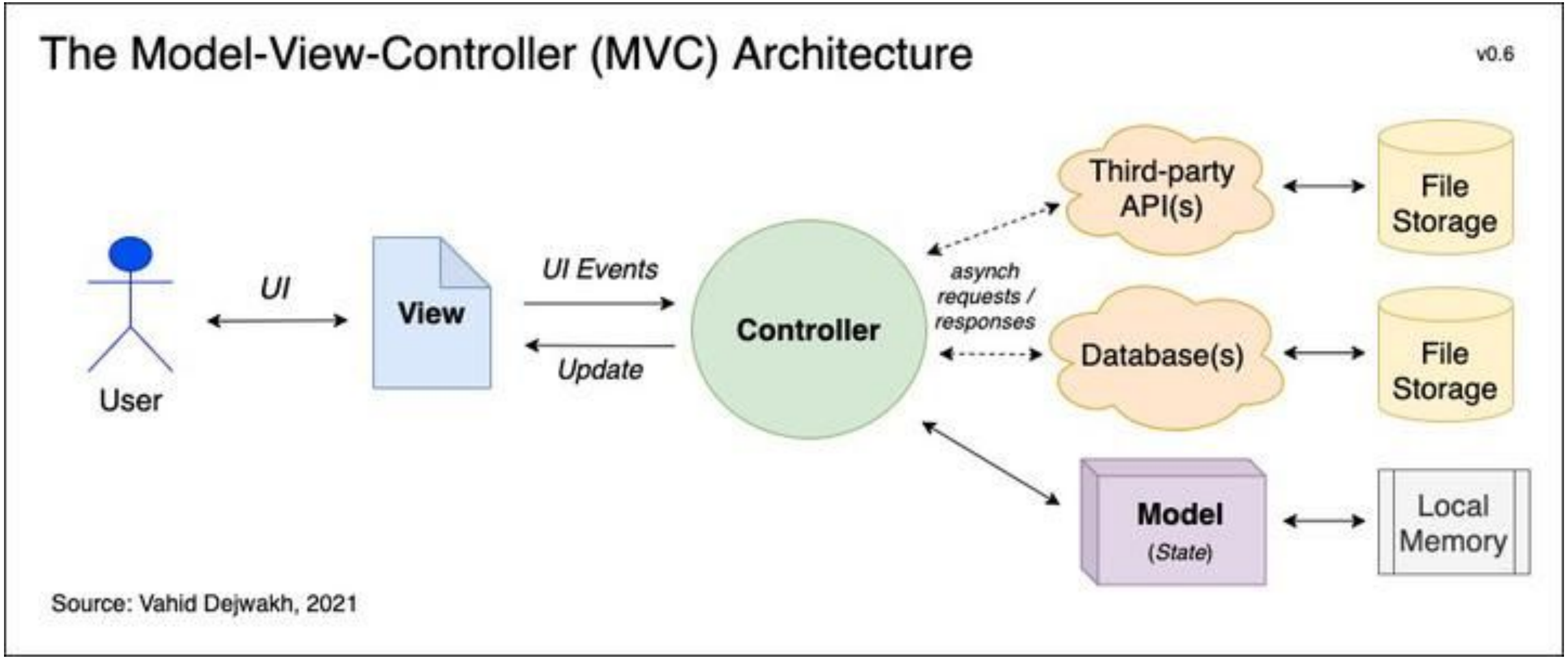
MVC: Controller

- The controller is the link/glue between the model and view.
- All communication between the model and the view happens through controller.
- For example, the `customer` controller will handle all the interactions and inputs from the `customer` view and update the database using the `customer` model.

MVC: Two representations



MVC: Another representation



MVC: Example

A typical use of an application that uses MVC, after the initial screen is rendered to the user is as follows:

1. The user triggers a view by clicking (typing, touching, and so on) a button
2. The view informs the controller of the user's action
3. The controller processes user input and interacts with the model
4. The model performs all the necessary validation and state changes and informs the controller about what should be done
5. The controller instructs the view to update and display the output appropriately

MVC: Why the Controller?

- Why the controller part? Can't we just skip it?
- We could, but then we would lose a big benefit that MVC provides:
    - The ability to use more than one view without modifying the model.
    - To achieve decoupling between the model and its representation, every view typically needs its own controller.
    - If the model communicated directly with a specific view, we wouldn't be able to use multiple views (or at least, not in a clean and modular way).

MVC: It’s everywhere

- MVC is a very generic and useful design pattern.
- Most popular web frameworks use it
  - Django, Rails, etc
- Most application frameworks use it
  - iPhone SDK, Android
- Or a variation: model-view-adapter (MVA), model-view-presenter (MVP)

MVC Benefits

- The separation between the view and model
    - Graphics designers focus on the UI part
    - Full-stack programmers focus on development of logic
- Because of the loose coupling between the view and model, each part can be modified/extended without affecting the other.
- For example, adding a new view is trivial.
    - Just implement a new controller for it.
- Maintaining each part is easier because the responsibilities are clear.

MVC: how to implement

When implementing MVC from scratch, be sure that you create:

- smart models
- thin controllers
- dumb views

MVC: Smart Models

- A model is considered smart because it does the following:
    - Contains all the data validation/business rules/logic
    - Handles the state of the application
    - Does not depend on the UI

MVC: Thin Controllers

- A controller is considered thin because it does the following:
    - Updates the model when the user interacts with the view
    - Updates the view when the model changes
    - Processes the data before delivering it to the model/view, if necessary
    - Does not display the data
    - Does not access the application data directly

MVC: Dumb Views

- A view is considered dumb because it does the following:
  - Displays the data
  - Allows the user to interact with it
  - Does only minimal processing, usually provided by a template language
    - (for example, using simple variables and loop controls)
  - Does not store any data
  - Does not access the application data directly
  - Does not contain validation/business rules/logic

MVC: Checking your instincts

- If you are implementing MVC from scratch and want to find out if you did it right, you can try answering some key questions:
    - How easily can you change the skin/look and feel of UI by changing just the view?
        - This should be easy with changes only to view
    - Is it easy to display the results in a new way (pie chart -> bar chart, etc
        - This should be either:
            - Changing just the view
            - Creating a new controller with a view attached to it, without modifying the model

MVC Example

Idea: Quote Printer.

The user enters a number and sees the quote related to that number. The quotes are stored in a quotes tuple. This is the data that normally exists in a database, file, and so on, and only the model has direct access to it.

MVC Example

```
quotes = (
     'As I said before, I never repeat myself.',
     'No free lunch',
     'Facts are stubborn things.'
      …
)
```

Example: Model

- The model is minimalistic in this case, since there are no updates to the quotes.
- It only has a get_quote() method
  - returns the quote (string) of the quotes tuple based on its index n.

Example: Model

```
class QuoteModel:

   def get_quote(self, n):

       try:

           value = quotes[n]

           return value

       except IndexError as err:

           raise ValueError('Invalid quote number: {}'.format(n)) from
err
```

Example: View

- The view has three methods:
    - show(), which is used to print a quote (or the message Not found!)
    - error(), which is used to print an error message on the screen
    - select_quote(), which reads the user's selection.

Example: View

```python
class QuoteTerminalView:

    def show(self, quote):

        print(f'And the quote is: "{quote}"')



    def error(self, msg):

        print(f'Error: {msg}')



    def select_quote(self):

        return input('Which quote number would you like to see? ')
```

Example: Controller

- The controller does the coordination.
- The __init__() method initializes the model and view.
- The run() method:
  - validates the quoted index given by the user
  - gets the quote from the model
  - passes it back to the view to be displayed as shown in the following code:

Example: Controller

```python
class QuoteTerminalController:
    def __init__(self):
        self.model = QuoteModel()
        self.view = QuoteTerminalView()

    def run(self):
        valid_input = False
        quote = None
        while not valid_input:
            try:
                n = self.view.select_quote()
                n = int(n)
                quote = self.model.get_quote(n)
                valid_input = True
            except Exception as e:
                self.view.error(f"Incorrect index '{n}'")
        self.view.show(quote)
```

Example: Main

The main() function initializes and fires the controller as shown in the following code:

```
def main():
        controller = QuoteTerminalController()
        while True:
             controller.run()
```

In-Class and Homework

- In small groups, do this tutorial for MVC with Django:
  - https://realpython.com/get-started-with-django-1/
- It’s ok if you don’t finish by end of class, but understanding this will help you a LOT with the projects.