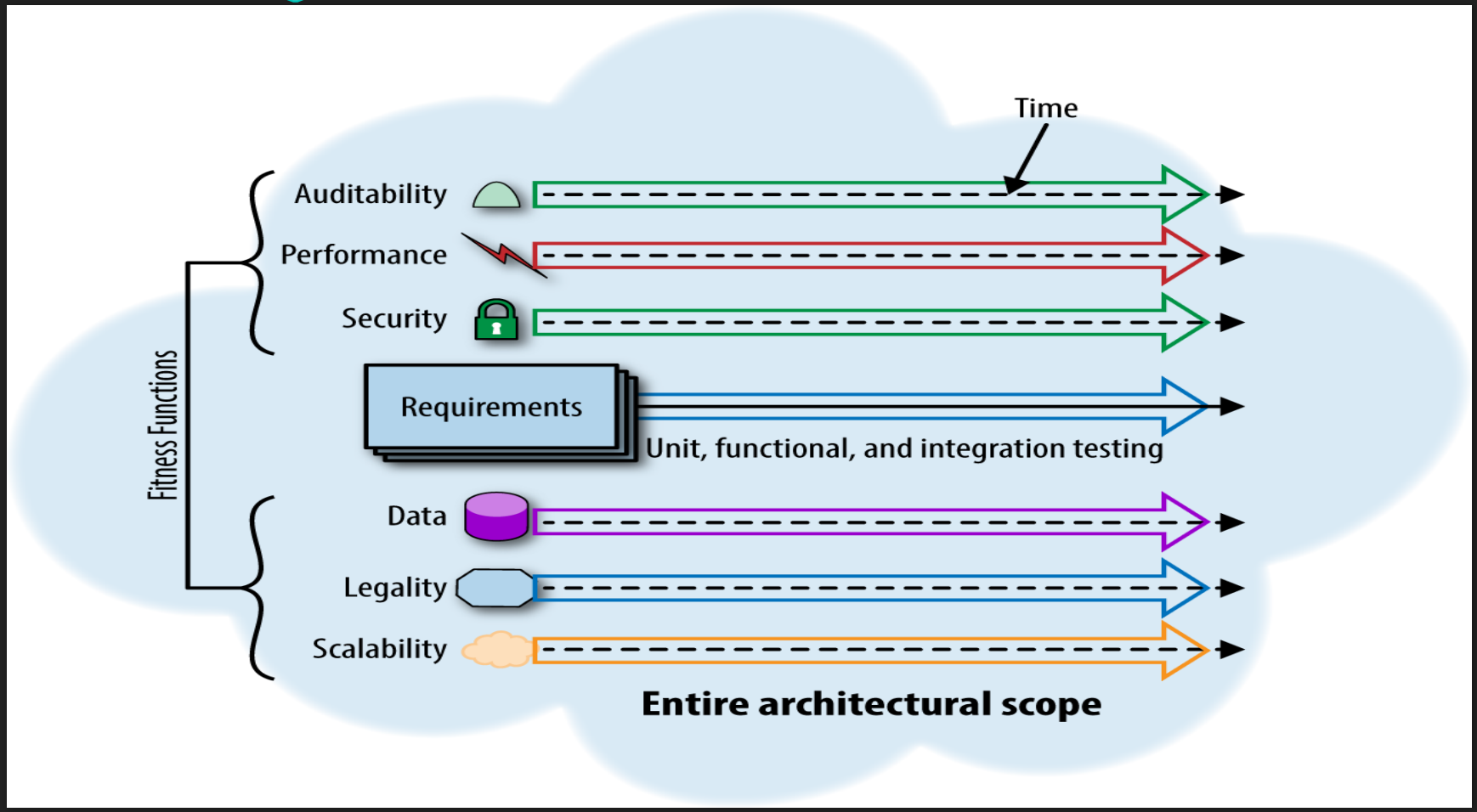
Software Architecture

most important aspects of the internal design of a software system

What is Architecture

- it's something like the fundamental organization of a system
- the way the highest level components are wired together.

Scope of Software Architecture



What Kinds of Architecture are there?

- Layered (n-tier) architecture
- Event-driven architecture
- Microkernel architecture
- Micro services architecture
- Space-based architecture

Layered (n-tier) architecture

- **Made for:** This approach is probably the most common because it is usually built around the database, and many applications in business naturally lend themselves to storing information in tables
- **Best for:**
  - New applications that need to be built quickly
  - Enterprise or business applications that need to mirror traditional IT departments and processes
  - Teams with inexperienced developers who don't understand other architectures yet
  - Applications requiring strict maintainability and testability standards
- Challenges:
  - New applications that need to be built quickly
  - Enterprise or business applications that need to mirror traditional IT departments and processes
  - Teams with inexperienced developers who don't understand other architectures yet
  - Applications requiring strict maintainability and testability standards

Event-driven architecture

- **Made for:** Many programs spend most of their time waiting for something to happen. This is especially true for computers that work directly with humans
- **Best for:**
  - Asynchronous systems with asynchronous data flow
  - Applications where the individual data blocks interact with only a few of the many modules
  - User interfaces
- Challenges:
  - Testing is hard and complex
  - Error handling is difficult
  - Messaging overload slow down the app
  - Developing a system wide data structure for events can be complex
  - Maintaining a transaction-based mechanism for consistency is difficult

Microkernel architecture

- **Made for:** Many applications have a core set of operations that are used again and again in different patterns that depend upon the data and the task at hand.
- **Best for:**
  - Tools used by a wide variety of people
  - Applications with a clear division between basic routines and higher order rules
  - Applications with a fixed set of core routines and a dynamic set of rules that must be updated frequently
- Challenges:
  - Deciding what belongs in the microkernel is often an art.
  - The plug-ins must include a fair amount of handshaking code so the microkernel is aware that the plug-in is installed and ready to work
  - Modifying the microkernel can be very difficult or even impossible once a number of plug-ins depend upon it
  - Choosing the right granularity for the kernel functions is difficult to do in advance

Micro services architecture

- **Made for:** The micro service architecture is designed to help developers avoid letting their modules grow up to be unwieldy, monolithic, and inflexible.
- **Best for:**
  - Tools used by a wide variety of people
  - Applications with a clear division between basic routines and higher order rules
  - Applications with a fixed set of core routines and a dynamic set of rules that must be updated frequently
- Challenges:
  - The services must be largely independent
  - Not all applications have tasks that can't be easily split into independent units
  - Performance can suffer when tasks are spread out between different micro services
  - Too many micro services can confuse users as parts of the web page appear much later than others

Space-based architecture

- **Made for:** The space-based architecture is designed to avoid functional collapse under high load
- **Best for:**
  - High-volume data like click streams and user logs
  - Low-value data that can be lost occasionally without big consequences—in other words, not bank transactions
  - Social networks
- Challenges:
  - Transactional support is more difficult with RAM databases
  - Generating enough load to test the system can be challenging, but the individual nodes can be tested independently
  - Developing the expertise to cache the data for speed without corrupting multiple copies is difficult

Which one is the best for us?

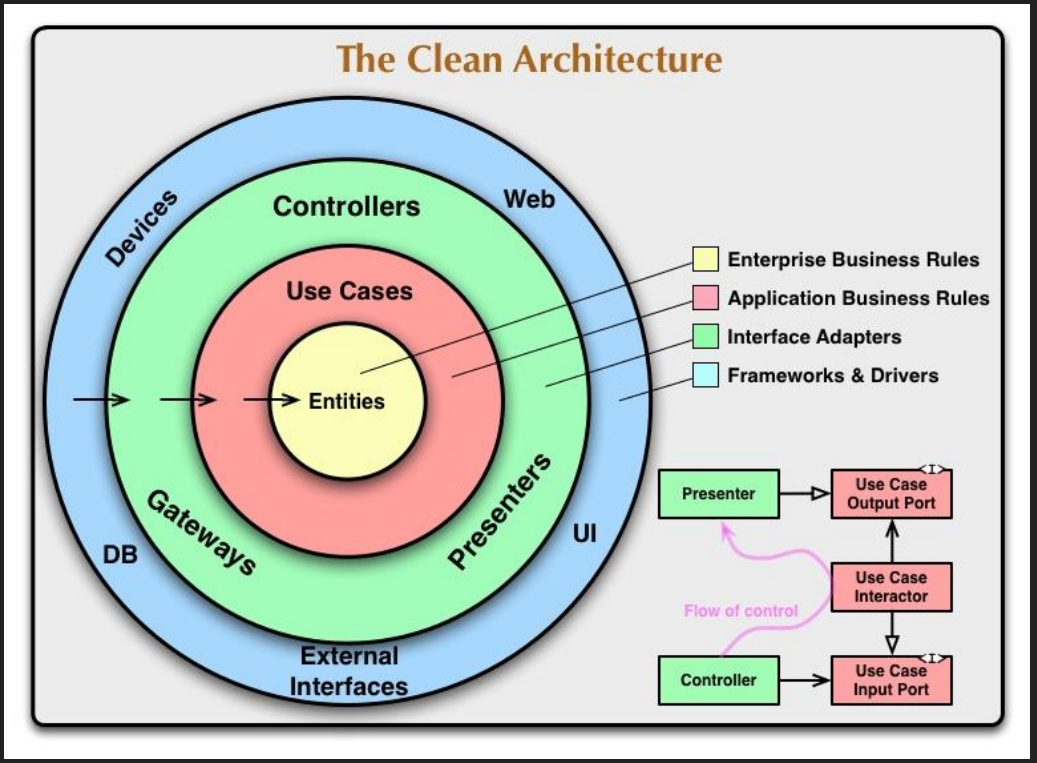
- Layered (n-tier) architecture is the best for us because it is:
  - More Maintainable
  - Testable
  - easy to assign roles
  - easy to update

More on Layered architecture

- The code is arranged so the data enters the top layer and works its way down each layer until it reaches the bottom, which is usually a database
- Along the way, each layer has a specific task, like checking the data for consistency or reformatting the values to keep them consistent
- It's common for different programmers to work independently on different layers.
- Proper layered architectures will have isolated layers that aren't affected by certain changes in other layers, allowing for easier refactoring.

What about Clean architecture?????

- "Clean Architecture" is more a "meta architecture", a high level guideline for creating layered architectures.



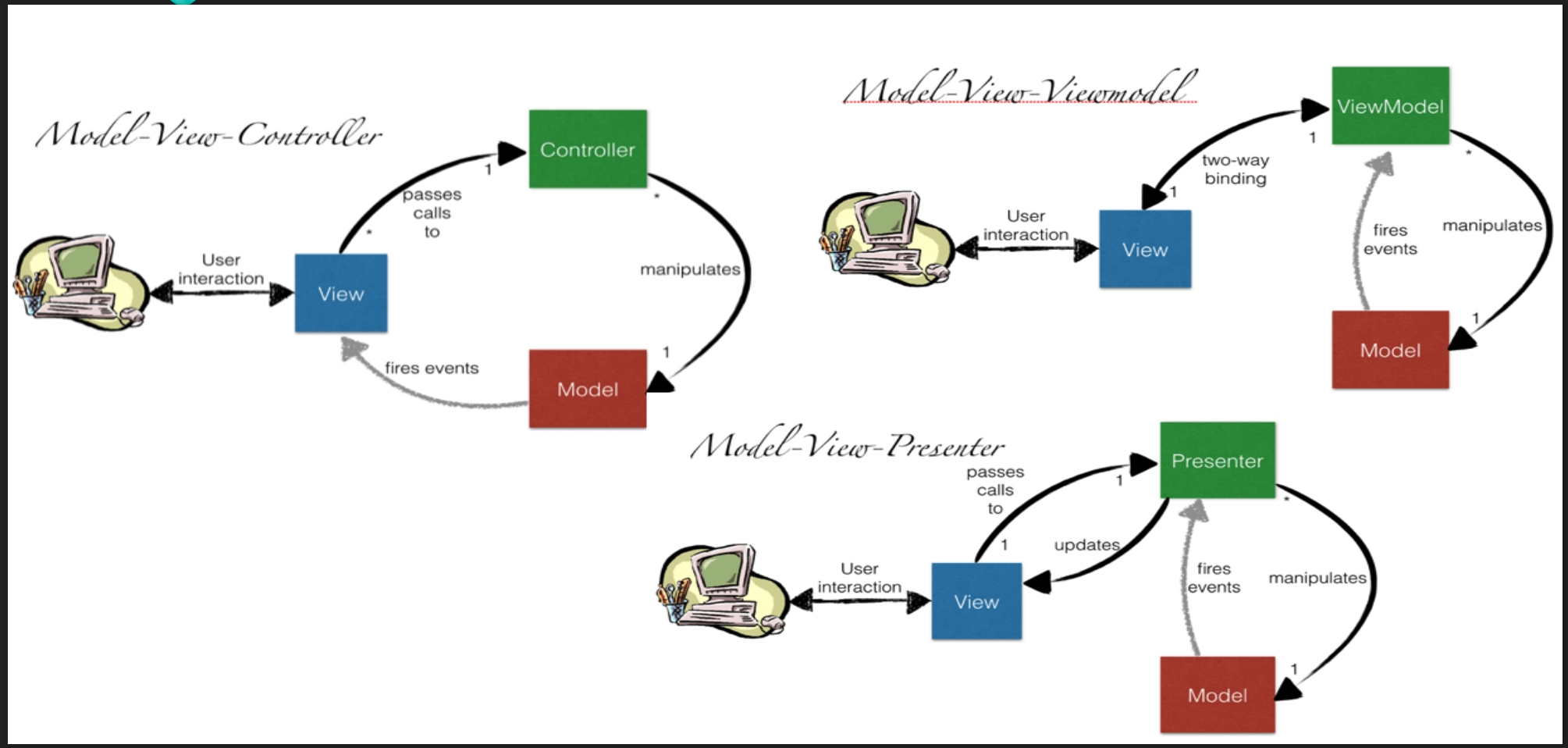
What's Clean architecture about?

- It is used for large projects
- It Makes them easy to change and grow
- separating the files or classes into components that can change independently
- SOLID principles:
  - **Single Responsibility Principle**
  - **Open Closed Principle**
  - **Liskov Substitution Principle**
  - **Interface Segregation Principle**
  - **Dependency Inversion Principle**

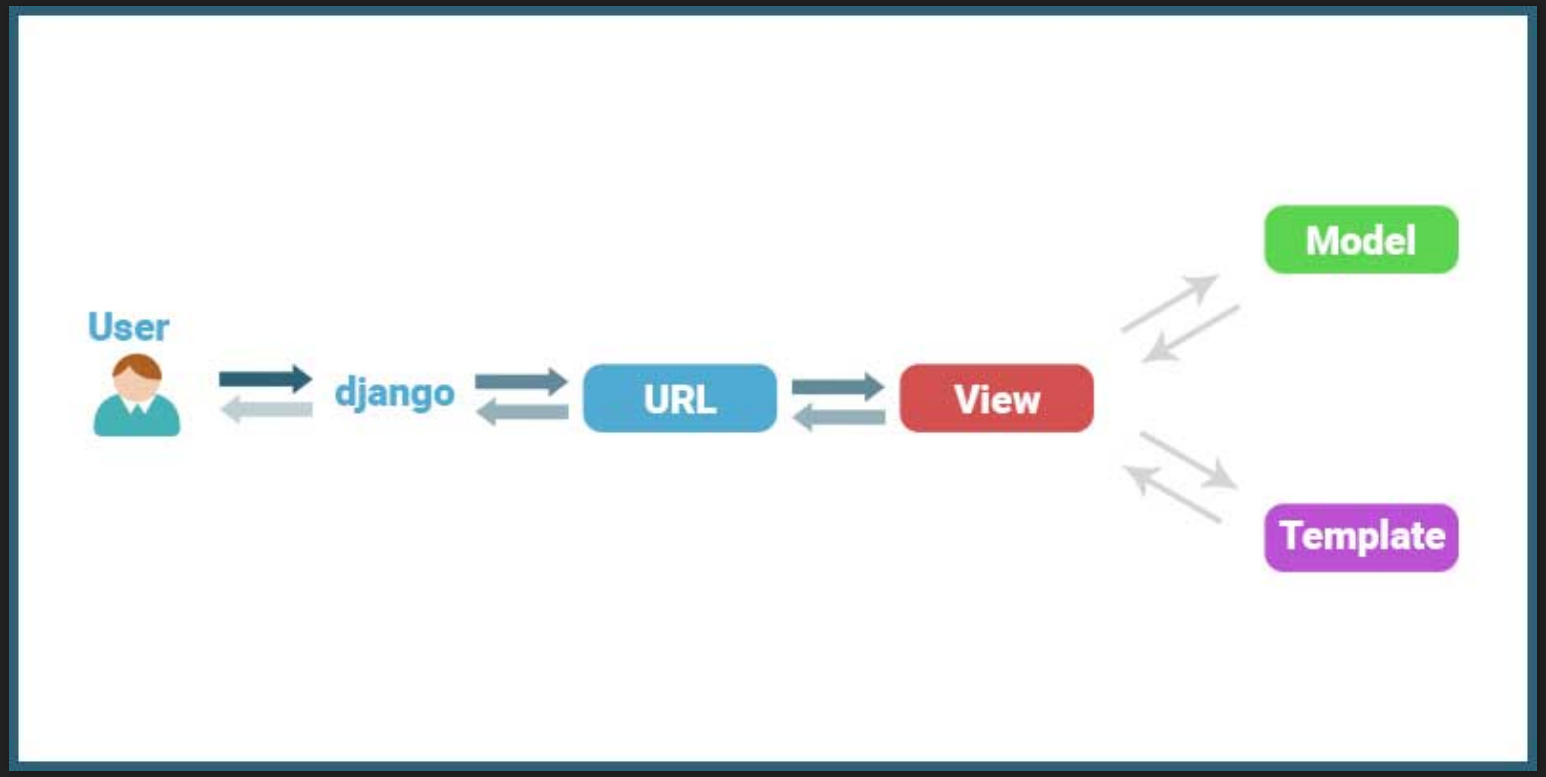
layered architecture types

- MVC (**Model–view–controller**)
- MVVM (**Model–view–viewmodel**)
- MVP (**Model–view–presenter**)
- PAC (**Presentation–abstraction–control**)
- MVT (**Model–view–Template**)
- MVI (**Model–view–Intent**)
- …

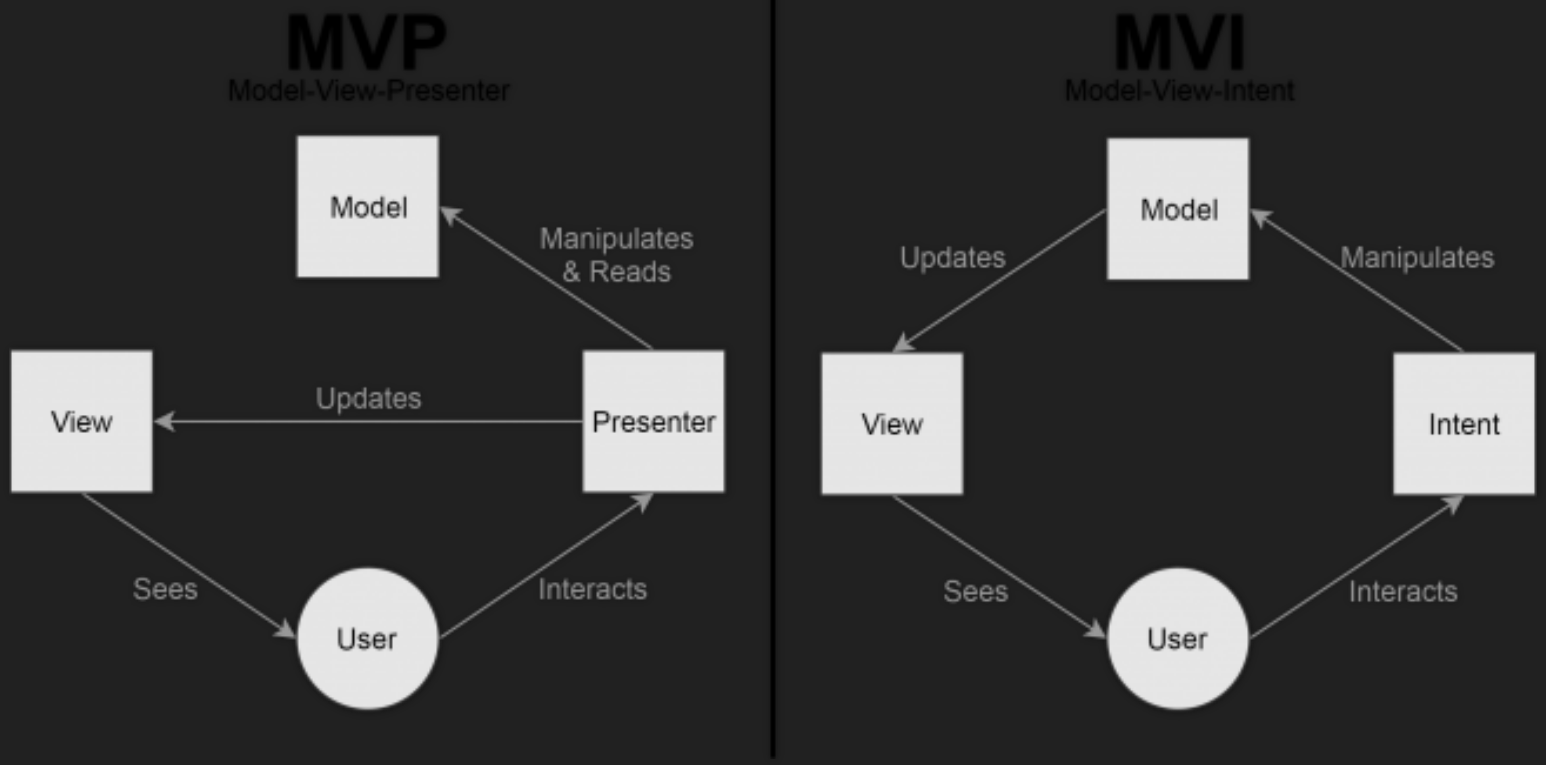
MVC vs MVVM vs MVP



MVT



MVI vs MVP


MVP
Model-View-Presenter
Model
Manipulates & Reads
View
Updates
Presenter
Sees
User
Interacts
MVI
Model-View-Intent
Model
Updates
Manipulates
View
Intent
Sees
User
Interacts

Which works better for us?

- Because we are using Django for the backend we have only two options:
  - MVC
  - MVT
- For the sake of Independence and separation of concerns, MVT it is.

The End.

- Thanks for paying attention
- Contact us on telegram:
  - @hamid_nem